UiO : Department of Physics
University of Oslo

# Wireless Embedded Microcontroller System for Bioimpedance Measurements

Master thesis

Patrick Hisni Brataas

**June 2, 2014**

# Contents

# Abstract

In this thesis the design and prototype of a bioimpedance measurement system is described. The system is wireless and battery powered. A custom analog front-end has been made to be able to do bioimpedance measurements. It uses a two or three electrode setup to do the measurements. It does most of the signal processing in the digital domain and two different techniques has been discussed. A common ARM microcontroller has been used to do the processing. The measurement data is transferred from the system to a desktop computer or Android device over Bluetooth.

The system has been tested and verified to work in the 1 kHz to 140 kHz frequency range. It measures the impedance modulus and phase correctly when tested with an electrical circuit. The egg white, egg yolk and the boundary between has been successfully discriminated by using a needle electrode as the measurement electrode.

# Acknowledgments

This thesis is the fullfilling of the Master of Science in Electronics and Computer Technology at the Department of Physics, University of Oslo. This work was carried out part-time from August 2012 to June 2014.

I would like to thank my supervisors Professor Ørjan G. Martinsen and Researcher Håvard Kalvøy for letting me undertake this project and for all feedback and help during the work on this thesis. A thanks to Oslo Bioimpedance Group for the trip to the International Conference on Electrial Bio-Impedance in Germany and the two day seminar at Sundvollen.

Thanks to the guys at the Electronics lab for all help.

It has been great being able to bounce ideas and ask questions to you Bent and thanks for the LaTeX template and valuable feedback.

At last I would like to thank my fantastic girlfriend Kristine for her infinite patience and understanding and my friends and family for their support during this time.

# Nomenclature

| | |
|---|---|
| $ADC$ | Analog to Digital Converter |
| $AFE$ | Analog Front-End |
| $AHB$ | Advanced High Speed Bus |
| $ANT$ | Apache Another Neat Tool |
| $APB$ | Advanced Peripheral Bus |
| $API$ | Application Programming Interface |
| $BIBO$ | Bounded-Input Bounded-Output |
| $BLM$ | Bilayer Lipid Membrane |
| $BMS$ | Bioimpedance Measurement System |
| $CooCox$ | Cooperate on Cortex |
| $CPE$ | Constant Phase Element |
| $DAC$ | Digital to Analog Converter |
| $DIP$ | Dual Inline Package |
| $DMA$ | Direct Memory Access |
| $DSP$ | Digital Signal Processing |
| $EA$ | Electrode Area |
| $EEA$ | Effective Electrode Area |
| $ENBW$ | Equivalent Noise Bandwidth |
| $ETS$ | Equivalent Time Sampling |

| | |
|---|---|
| *FEM* | Finite Element Method |
| *FFT* | Fast Fourier Transform |
| *FIFO* | First-In First-Out |
| *FIR* | Finite Impulse Response |
| *FPU* | Floating Point Unit |
| *GBP* | Gain Bandwidth Product |
| *GCC* | GNU Compiler Collection |
| *GDB* | GNU Debugger |
| *GNU* | GNU's Not Unix |
| *GUI* | Graphical user interface |
| *HSE* | High Speed External |
| *HSI* | High Speed Internal |
| *I2C* | Inter-Integrated Circuit |
| *IC* | Integrated Circuit |
| *IDE* | Integrated Development Environment |
| *IIR* | Infinite Impulse Response |
| *IRQ* | Interrupt Request |
| *ISR* | Interrupt Service Routine |
| *IWDG* | Independent Watchdog |
| *JDK* | Java Development Kit |
| *JTAG* | Joint Test Action Group |
| *MinGW* | Minimalist GNU for Windows |
| *NDK* | Native Development Kit |
| *PCB* | Printed Circuit Board |

| | |
|---|---|
| $PID$ | Proportional Integral Derivative |
| $PLL$ | Phase Locked Loop |
| $QML$ | Qt Meta/Modeling Language |
| $RC$ | Resistor Capacitor |
| $RISC$ | Reduced Instruction Set Computer |
| $RMS$ | Root Mean Square |
| $ROM$ | Read Only Memory |
| $RSS$ | Root Sum Square |
| $RTC$ | Real Time Clock |
| $RTOS$ | Real Time Operating System |
| $SDK$ | Software Development Kit |
| $SNR$ | Signal-to-Noise Ratio |
| $SPL$ | Standard Peripheral Library |
| $TIA$ | Transimpedance Amplifier |
| $ZCD$ | Zero Crossing Detector |

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background and Motivation

Commercial instruments for measuring bioimpedance is often large and expensive. A small, lightweight and cheap device with a more specific use in mind is in many cases more desirable. If it were to be used for a single or a few clinical application the device can be optimized for this use.

Designing a platform from scratch ensures it is very flexible and can serve as a platform for future development. The limits for what can be achieved with microcontrollers are continuously pushed and this makes it possible to implement more functionality.

Using a computer, tablet or mobile device to visualize data is becoming more and more popular, especially the two latter.

## 1.2 Goal of this Thesis

It was desired to design a small wireless and battery powered device to measure bioimpedance in biological materials. The device should do the processing in the digital domain on a microcontroller, and send the data wirelessly to a connected device for visualization. The ultimate goal was to be able to show that this system could successfully discriminate between different biological materials.

The goal can be divided into three parts:

1. Develop a microcontroller system that successfully measures impedance.

2. Develop software that wirelessly receive the impedance data and visualizes it on a display.

3. Successfully discriminate between some biological tissues using a needle measurement electrode.

# Chapter 2

# Bioimpedance Theory

This chapter gives an introduction to basic bioimpedance theory.

## 2.1 Basics

### 2.1.1 Resistor

A resistor is simply a passive electrical component that reduces current flow and indirectly the voltage. This is because the current and voltage are directly proportional in a resistor. This leads to Ohm's law:

$$V = RI \tag{2.1}$$

where V is the voltage across the resistor, R is the resistance of the resistor and I the current flowing through.

### 2.1.2 Capacitor

A capacitor is comprised of two conductors that is separated by air or some kind of material. The material separating the two conductors is called a dielectric. When a voltage is applied over a capacitor, it will charge and the relationship between how much it charges and the voltage is called capacitance. Its equation is usually written:

$$C = \frac{Q}{V} \tag{2.2}$$

where C is the capacitance, Q is the charge and V the voltage. The type of dielectric and the geometry of the capacitor determines the capacitance.

### 2.1.3 Bode plot

A bode plot shows the magnitude response and phase as a function of frequency. The x-axis is frequency in logarithmic scale. This plot gives a good overview of how magnitude and phase varies with frequency.

### 2.1.4 Wessel plot

A Wessel plot displays the real and imaginary part of the immittance as the x-axis and y-axis respectively. This if often seen in the field of bioimpedance when comparing and fitting experimental results with the Cole model.

## 2.2 Immittance

Immittance is a more general term meaning impedance or admittance for those cases where differentiating between the two has no value. Since Immittance is just a term, it does not have its own unit.

### 2.2.1 Impedance

Before we can address bioimpedance, we must know what impedance is. In this thesis it is implied that when using the word impedance it is referring to electrical impedance. Impedance is measured in Ohms [Ω] and is a measure of how well, for example, an electrical circuit oppose the resulting alternating current when an alternating voltage is applied to it. This ratio can be written as:

$$Z = \frac{v}{i} \tag{2.3}$$

where v is the applied voltage and i is the current. Written in the Cartesian form, impedance consists of two parts:

$$Z = R + jX \tag{2.4}$$

where the real part R is the resistance and the imaginary part X is the reactance. If written in complex exponential form:

$$Z = |Z|e^{j\theta} \tag{2.5}$$

|Z| is the magnitude and $\theta$ the phase difference between the voltage and current in the time domain. These two forms express the same and the calculation from one form to the other is trivial:

$$|Z| = \sqrt{R^2 + X^2} \tag{2.6}$$

$$\theta = arctan(\frac{X}{R}) \tag{2.7}$$

$$R = |Z|cos(\theta) \tag{2.8}$$

$$X = |Z|sin(\theta) \tag{2.9}$$

### 2.2.2 Admittance

Admittance is measured in Siemens [S] and is the inverse of impedance. This is a measure of how well, for example, an electrical circuit conducts current. The admittance can be measured when applying an alternating current and measuring the resulting alternating voltage. This results in the ratio:

$$Y = \frac{1}{Z} = \frac{i}{v} \tag{2.10}$$

where i is the applied current and v the voltage. Written in Cartesian form we get:

$$Y = G + jB \tag{2.11}$$

where the real part G is the conductance and the imaginary part B is the susceptance. In addition, like impedance, admittance can be written on the complex exponential form. By using complex numbers theory it can be shown that it is possible to convert to impedance to admittance and vice versa. The resulting equations for the conversions are:

Impedance to admittance:

$$G = \frac{R}{R^2 + X^2} \tag{2.12}$$

$$B = -\frac{X}{R^2 + X^2} \tag{2.13}$$

Admittance to impedance:

$$R = \frac{G}{G^2 + B^2} \tag{2.14}$$

$$X = -\frac{B}{G^2 + B^2} \tag{2.15}$$

### 2.2.3 Resistivity, conductivity, impedivity, admittivity, immittivity and permittivity

Resistance is dependent on both the materials resistivity and geometry:

$$R = \rho G \tag{2.16}$$

where R is the resistance, $\rho$ the resistivity and G the geometry. In for example a cylindrical resistor the resistance is given as:

$$R = \rho \frac{l}{A} \tag{2.17}$$

where l is the length of the cylinder and A is the cross-sectional area.

A similar dependency is found for a parallel plate capacitors capacitance and permittivity:

$$C = \epsilon G = \epsilon \frac{A}{d} \tag{2.18}$$

where $C$ is the capacitance, $\epsilon$ is the permittivity, $A$ the plate area and $d$ is the distance between the plates.

This means the –ance suffix parameters are dependent on both the electrical properties of the material and the geometry, while the –ivity suffix implies parameters that only depend on the electrical properties of the material itself, not the geometry. (Grimnes and Martinsen, 2008, p. 2)

## 2.3 Bioimpedance

Bioimpedance is the term used when referring to the impedance of a biological substance. Bioimpedance describes the passive electrical properties of biological materials. Unlike metals where the charge carriers are the free electrons, in biological materials the charge carriers are the ions in the intracellular and extra-cellular liquids. The most important ions are K+ and Na+ as can be seen in table 2.1.

Since ions are the charge carriers, a current flow implies a movement of substance, which in itself results in changes in the biological material. Hence, a DC current would change the conductor itself, first at the electrodes, but over time in the whole material. (Grimnes and Martinsen, 2008, p. 8))

### 2.3.1 Electrical properties of tissue

The cell membrane is a poor electrical conductor due to phospholipids forming a bilayer lipid membrane (BLM) and acts as a dielectric, which means

| Cations (meq/L) | | | Anions (meq/L) | | |
|---|---|---|---|---|---|
| | **Plasma** | **Intracellular** | | **Plasma** | **Intracellular** |
| $Na^+$ | 142 | 10 | $Cl^-$ | 103 | 4 |
| $K^+$ | 4 | 140 | $HCO_3^-$ | 24 | 10 |
| $Ca^{2+}$ | 5 | $10^{-4}$ | $Protein^-$ | 16 | 36 |
| $Mg^{2+}$ | 2 | 30 | $HPO_4^{2-} + SO_4^{2-}$ + organic acids | 10 | 130 |
| $H^+$ (pH = 7.4) | $4 \times 10^{-5}$ | $4 \times 10^{-5}$ | | | |
| Sum | 153 | 180 | Sum | 153 | 180 |

0.9% NaCl is 154 mmol/L.

Figure 2.1: Concentration of electrolytes in intra-cellular and extra-cellular liquids in milliequivalents per liter. Taken from (Grimnes and Martinsen, 2008, p. 24)

the cells have capacitive properties. The conductivity of the membrane itself is very low and the membrane is only about 7 nm thick. Because of this, the cell has a very high capacitance of about 1 $\frac{\mu F}{cm^2}$. (Grimnes and Martinsen, 2008, p. 100) Due to the diffusion of ions through the cell membrane, the concentration of ions in the intracellular and extracellular liquids can change. The migration of ions through the membrane is possible due to ion channels. These channels are ion specific.

At DC and low frequencies the current flows around the cells in the extracellular liquids, at higher frequencies the cell membrane capacitance allows AC current to pass through. For frequencies in between there will be a mix of how much goes around and through the cell. Approximate values for low, medium and high frequencies in this context is less than 10 kHz, 100 kHz and above 1 MHz respectively. This is shown in figure 2.2. Biological tissue also have inductive properties, but these are insignificant at frequencies below 10 MHz (Riu, 2004) and not further discussed in this thesis.

In general, the impedance of tissue decreases with an increase in frequency.

Biological tissue is a very heterogeneous material and different tissues like muscles, fat, organs, body fluids etc. can have very different cell structures. This includes cell size and shape, how they are joined together and how much e.g. extracellular liquid that is present in a specific tissue. Adipose tissue has little extracellular electrolytic liquid for conduction of lower frequencies while skeletal muscles might have anisotropic (direction dependent) properties.

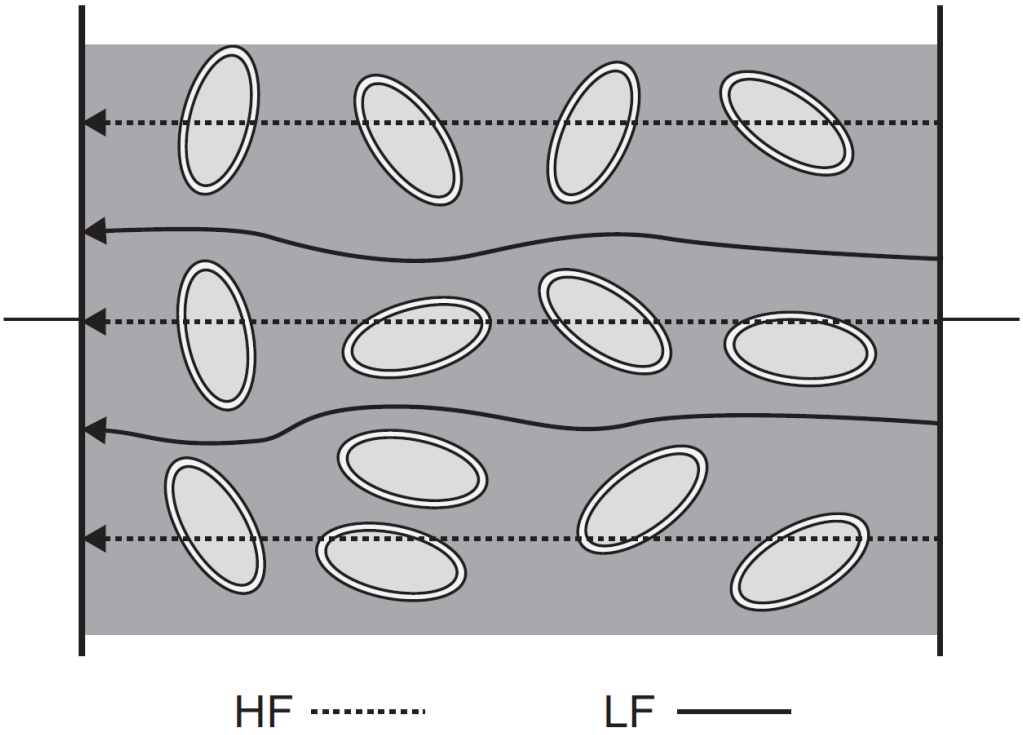



Figure 2.2: At low frequencies (LF), the extracellular path dominates due to the capacitive properties of the cell membrane. At high frequencies (HF) the cells capacitive properties allows the AC current to pass through. Taken from (Grimnes and Martinsen, 2008, p. 103)

### 2.3.2 Polarization

Polarization is the disturbance in the charge distribution in a region as a result of an induced electric field. This means the charge does not change while the distribution of charge does. Since transport of ionic current also means that substance will be moved, there will be concentration changes in the biological material near the electrodes. In the field of bioimpedance, the measuring methods used are exogenic. This means energy is applied to a system, which will polarize it. The energy applied can be both be stored and dissipated in a dielectric medium. All materials are polarizable, altough to different degrees. This includes conductors and insulators. An electric dipole is a separation of positive and negative charges. An example of a dipole is a water molecule which has a negative and positive pole. Dielectric theories regarding biomaterials often consider them as either polar materials or as inhomogeneous materials with interfacial polarization contributions. (Grimnes and Martinsen, 2008, p. 58) The polarization is the sum of three components:

1. Number of induced dipoles due to an applied external field.

2. Orientation of the already existing permanent dipoles in the direction of an applied electric field.

3. The permanent dipole moment when no electric field is present.

There is also an important difference between electronic polarization and ionic polarization. Electronic polarization is a displacement of the electron cloud with respect to the nucleus (protons). Ionic polarization is the displacement of positive ions with respect to the negative ions.

Polarization of biological materials does not occur instantaneously. When applying an AC signal the frequency is critical for how much polarization we will observe, depending on the time the charges need to change their positions in the material. If the frequency is low enough it will mean that all charges have enough time to change their position and the polarization will be maximal. For the same reason, polarization will decrease with increasing frequency. The time dipole molecules need to reorient themselves is called relaxation time and has a relaxation time constant. This can be seen in the time domain. In the frequency domain when plotting permittivity as a function of frequency, one can see the effect of relaxation. This characteristic is called dispersion.

### 2.3.3 Dispersion

Frequency dispersions can be found when measuring on biological materials. Dielectric dispersion is the dependency between the permittivity and frequency in a material when we polarize it. There is also a lag between the polarization and the changes in the applied electrical field. Three dispersion areas were described by Schwan in 1957 and named $\alpha$, $\beta$ and $\gamma$. Later a fourth dispersion area between $\beta$ and $\gamma$ has been described and named $\delta$. The typical idealized dispersion regions for tissue can be shown in figure 2.3. (Martinsen et al., 2002) Different tissue have different degrees of dispersion. For example, muscle tissue exhibits a large $\alpha$-dispersion while blood does not have one.

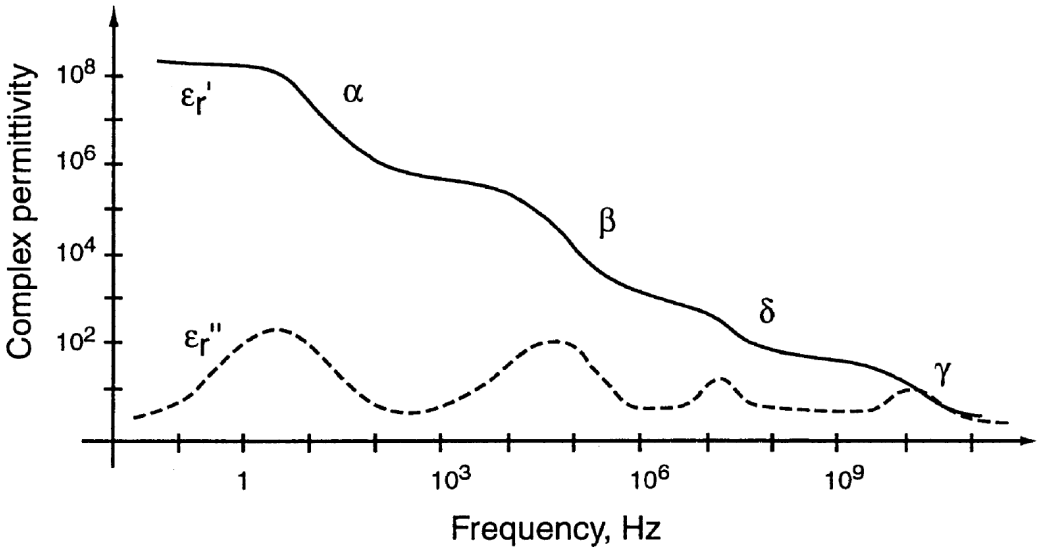



Figure 2.3: Idealized dispersion regions for tissue. DC conductance have been subtracted from the imaginary part of the complex permittivity $\epsilon_r''$ value. Taken from (Martinsen et al., 2002)

Many mechanisms contribute to these dispersions and in table 2.4 one can see the approximate frequency regions for the different dispersions and what mechanisms responsible are.

### 2.3.4 Equivalent electrical circuit for biological tissue and the Cole model

The cell bioimpedance and tissue bioimpedance can be modeled by using an equivalent electrical circuit. Many of these models exists, but one of the most

| Type | Characteristic frequency | Mechanism |
|---|---|---|
| $\alpha$ | mHz–kHz | Counterion effects (perpendicular or lateral) near the membrane surfaces, active cell membrane effects and gated channels, intracellular structures (e.g. sarcotubular system.), ionic diffusion, dielectric losses (at lower frequencies the lower the conductivity). |
| $\beta$ | 0.001–100 MHz | Maxwell–Wagner effects, passive cell membrane capacitance, intracellular organelle membranes, protein molecule response. |
| $\gamma$ | 0.1–100 GHz | Dipolar mechanisms in polar media such as water, salts and proteins. |

Figure 2.4: The frequency regions and mechanisms that corresponds to the different dispersions. Taken from (Grimnes and Martinsen, 2008, p. 90)

common model is displayed in figure 2.5.

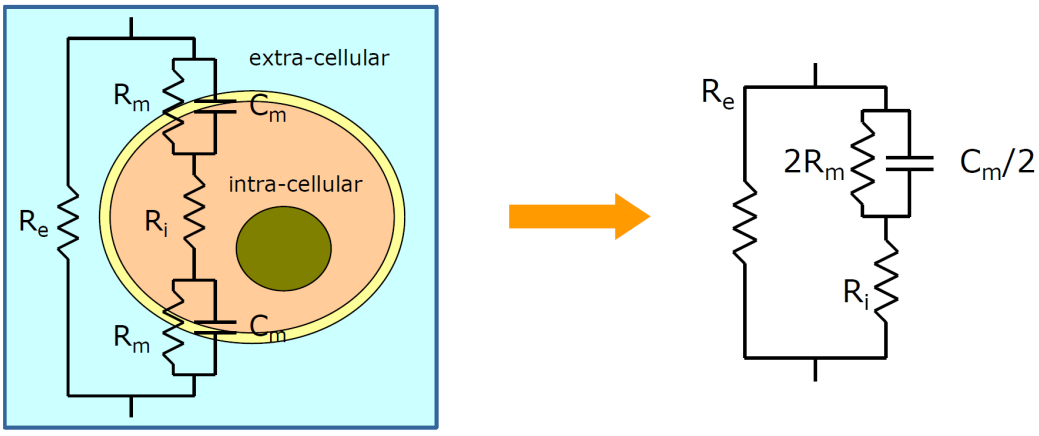



Figure 2.5: Electrical model of biological material. Taken from (Ivorra, 2003)

The circuit on the right in figure 2.5 is equivalent to the left after some simplifications. (Ivorra, 2003) As shown, the current can flow through the extracellular liquid ($R_e$), through the bilayer lipid membrane ($C_m$) or across the ionic channels ($R_m$). Once the current has entered the cell it also travels through the intracellular liquid ($R_i$) and across the cell membrane ($R_m||C_m$). $R_m$ is often omitted as the cell membrane is a poor conductor. This model is also often seen as an equivalent model in the literature. This model only has a single dispersion, but tissue often have more than one dispersion overlapping in some of the frequency area. This requires a more complex model where the capacitor in the previous model is substituted by a Constant Phase Element

(CPE). The CPE is not an electronic component that exists, but is described as a frequency dependent capacitor and resistor. It can be modeled so that the phase is frequency independent. (Grimnes and Martinsen, 2008, p. 299) It is also called a fractional capacitor. CPE has the impedance:

$$Z_{CPE} = \frac{1}{(j2\pi fC)^{\alpha}} \tag{2.19}$$

where j is the imaginary unit, f the frequency, C the capacitance and $\alpha$ a parameter with a value usually between 0.5 and 1. When $\alpha$ =1 the CPE behaves identical to an ideal capacitor and when $\alpha$ =0 it behaves like an ideal resistor. When we substitute $R_m$ and $C_m$ with CPE in the circuit in figure 2.5 we get the impedance expressed as:

$$Z = R_{\infty} + \frac{R_0 - R_{\infty}}{1 + (j2\pi f\tau)^{\alpha}} \tag{2.20}$$

This equation is called the Cole equation. (Cole, 1940) $R_{\infty} = R_i$ is the resistive part at infinite frequency, $R_0$ is the impedance at 0 Hz, $\tau$ is the time constant $(\Delta RC)^{\frac{1}{\alpha}}$ (Elwakil and Maundy, 2010) where $\Delta R = R_m = R_0 - R_{\infty}$ and $\alpha$ the CPE parameter.

Different tissues can be characterized by finding these four parameters. These four parameters can be found by measuring with an impedance analyzer and plotting the values in a Wessel plot, like in figure 2.6.

The real part in figure 2.6 is the resistance and the imaginary part the reactance. $R_0$ and $R_{\infty}$ can be found where the arc intersects the $Z^{'}$-axis. The CPE angle can be found as the angle between the tangent along the arc at $R_{\infty}$ and the R-axis. $\alpha$ can now be calculated as:

$$\alpha = \frac{2\varphi_{CPE}}{\pi} \tag{2.21}$$

$\tau$ can be found by finding the frequency where $|X|$ has its maximum value. $\tau$ is then calculated as:

$$\tau = \frac{1}{2\pi f_{|X|max}} \tag{2.22}$$

When more than one dispersion occurs, the Cole equation can be expanded:

$$Z = R_{\infty} + \frac{(R_0 - R_{\infty})_1}{1 + (j2\pi f\tau_1)^{\alpha_1}} + \frac{(R_0 - R_{\infty})_2}{1 + (j2\pi f\tau_2)^{\alpha_2}} + ... \tag{2.23}$$

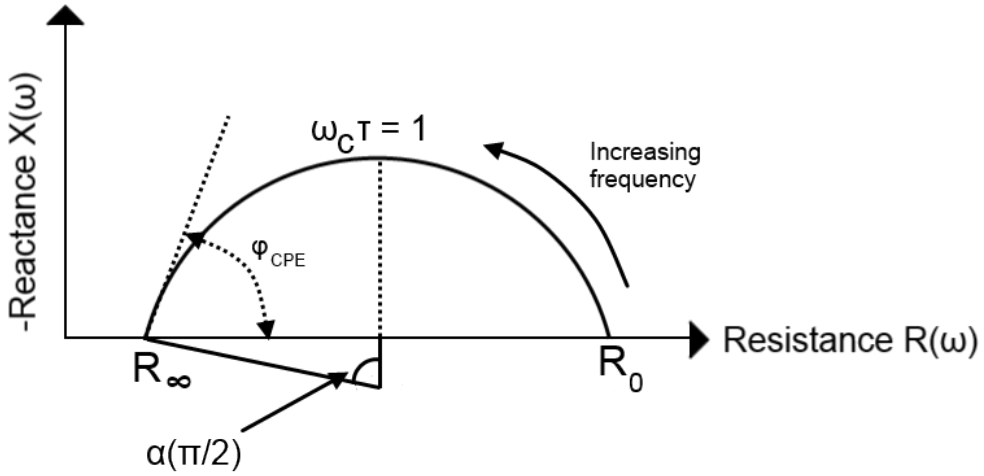



Figure 2.6: Wessel plot. Notice that the reactance-axis have been inversed. This is often seen in Wessel plots. Redrawn from (Nordbotten, 2008, p. 11) and modified.

## 2.4 Electrodes and Measurement

To be able to measure bioimpedance an interface between the biological sample and the electronics is needed. Electrodes provide this interface. At the electrode is where the shift in charge carriers occur. Between the free flowing electrons in the metal to the ions in the biological material and vice versa. This means the electrode is a transducer, converting a physical quantity (ions) into an electric signal. A pair of electrodes can be used to carry current, measuring a potential difference with no current flowing or both. (Grimnes and Martinsen, 2008, p. 253) At the interface between the metal and electrolyte, the AC current flow is impeded to some degree and this creates a polarization impedance. Therefore, when current flow in the electrode wire the electrode is polarized. There are many ways to design and configure the electrodes. This is in many applications critical for correct measurements. The most common electrode configurations are 2-, 3- and 4- electrode setups.

### 2.4.1 Monopolar measurements

When using two or more electrodes one can achieve monopolar measurements by making one of the electrodes dominant. This can for example be done in a two-electrode setup by increasing or decreasing the size of one of the electrodes relative to the other. This can be seen in figure 2.7 where figure 2.7a

shows two equal electrodes in a symmetrical bipolar system and figure 2.7b shows a quasi-monopolar system. In figure 2.7b the sensitivity and therefore the impedance contribution is closer to the small measuring electrode than in figure 2.7a and the impedance contribution from the large electrode would be insignificant.

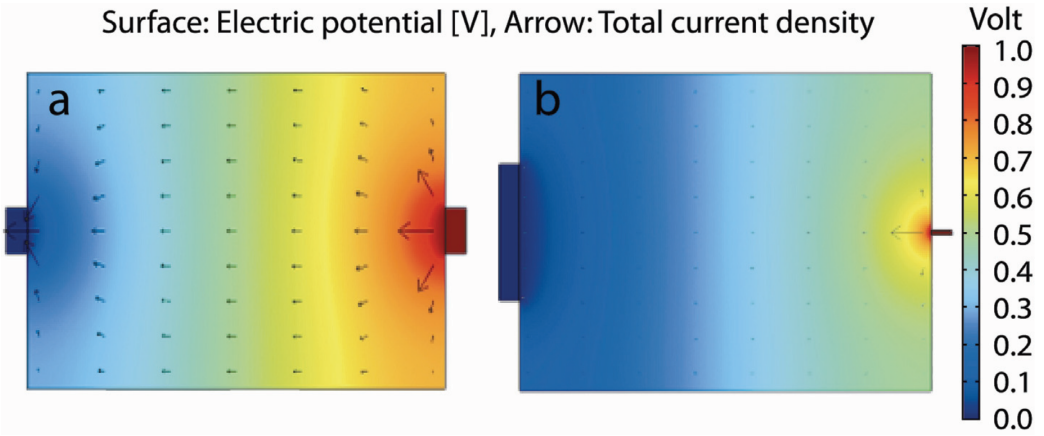



Figure 2.7: The arrows shows the current density and the color the potential distribution in this two-electrode configuration. a) Equal and symmetrical electrodes. b) Quasi-monopolar setup with non-symmetrical electrode configuration. Taken from (Kalvoy, 2010)

### 2.4.2 Two electrode setup

In a two-electrode setup, the same pair of electrodes are used to both excite and measure. The immittance of the system is measured and includes the whole setup. This means the electrodes, electrode interfaces, leads and the sample to be measured are all connected in series and will affect the measured immittance. The impedance can be measured by applying a controlled current signal and measuring the voltage and vice versa for admittance.

### 2.4.3 Four electrode setup

A four electrode setup uses two pairs of electrodes and is a two-port network with four terminals. One pair for current carrying and one pair for pick up. This can be seen in figure 2.8 where CC1 and CC2 is the current carrying electrodes and PU1 and PU2 the pick-up electrodes. The pick-up electrodes are usually connected to a very high impedance operational amplifier, which means in practice that no current will flow past these electrodes. When no

current flows through an electrode it will not be polarized. This means that when using the four electrode setup the electrode polarization impedance will not be affecting the measurements. This setup is popular, but also has its pitfalls and are more vulnerable to errors than monopolar and bipolar setup. (Grimnes and Martinsen, 2006) One of the pitfalls is zones of negative sensitivity. This means that when measuring on heterogeneous materials it is possible that the measurements can have large errors. If the electrodes are placed as shown in figure 2.8 it is often believed that only the volume between the pick-up electrodes are measured. This is not true as the area between the pick-up electrode and the current-carrying electrode will have negative sensitivity. If the impedivity is increased in this area the total measured impedance will be lower and might not give the expected result. The negative sensitivity area of a four-electrode setup can be seen in a finite element method (FEM) simulation where the measured material is infinite and homogenous. This is shown can be seen in figure 2.9.

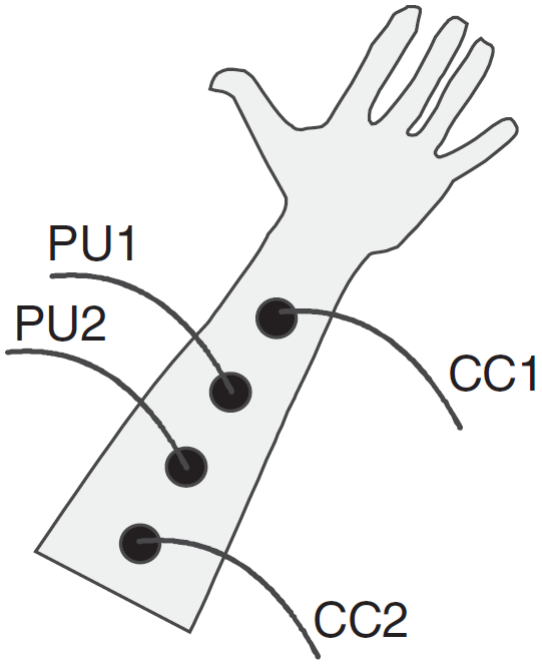



Figure 2.8: Four-electrode configuration on the underarm with the current carrying electrodes placed as outer electrodes and pick-up electrodes as inner electrodes. Taken from (Grimnes and Martinsen, 2006)

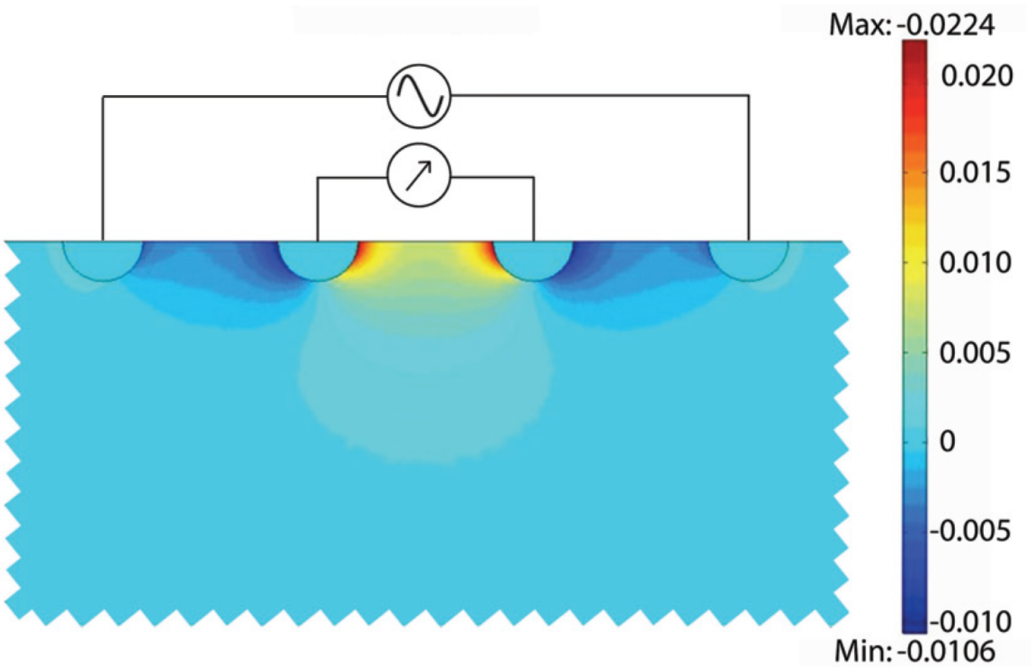



Figure 2.9: Shows the sensitivity area for a four-electrode setup in an infinite homogenous material. The dark blue indicate areas with negative sensitivity. Taken from (Kalvoy, 2010)

### 2.4.4 Transfer immittance

It is important to be aware that a three and four electrode setup measures transfer immittance. This is because they are both a two-port network. For example if an electric current excitation signal is applied and the response is a voltage, we measure transfer impedance. Visa versa for transfer admittance. Transfer immittance distinguishes itself from immittance as it shows how much of the applied signal is sensed at the measuring electrodes. When we apply a current excitation signal, increasing the distance between the measuring electrodes and the electrodes where the current excitation signal is applied, will result in a smaller potential difference and impedance. When moved infinitely far apart the material in between seems to have zero resistance. This is clearly not correct and this behavior is very important to be aware of when using these kind of electrode configurations. (Martinsen and Grimnes, 2008) So it is not actually the impedance of the material itself that is measured and finding resistivity, conductivity and permittivity is practically not possible in an inhomogeneous material. Figure 2.8 shows the current carrying electrodes as the inner electrodes and the pick-up electrodes as the outer electrodes. It does not matter if they are interchanged as the reciprocity theorem guarantees that the same immittance will be measured under linear conditions. (Grimnes and Martinsen, 2006)

### 2.4.5 Three electrode setup

A three electrode setup uses three electrodes where two are current carrying electrodes and one is a pick-up electrode. This means it is a two-port network with three terminals. The current carrying electrodes are shown as M and C in figure 2.10. The pick-up electrode is electrode R. The measuring electrode M measures the potential between itself and the reference electrode R which is guaranteed to have the same potential as the excitation voltage.

Because the input impedance of the operational amplifier is very high, no current flows through the reference electrode. This means its contribution to the measurement is very small and no electrode polarization impedance since no current flow. This property makes it easier make a monopolar system by making the measuring electrode smaller, since the size of the reference and current carrying electrode C is less crucial. In this monopolar three-electrode setup the transfer immittance between the measurement electrode and the reference is measured and most of the contribution is from areas close to the measurement electrode. The electrode polarization impedance from electrode M is also included in the measurement. The monopolar characteristics can be seen in figure 2.10 even when the electrodes are of equal size and shape.

The circuit in figure 2.10 does remove the 50/60 Hz mains noise that the material measured on picks up, as the current carrying electrode C and the operational amplifier it is connected to will compensate for it, without interfering with the measuring electrode.

If swapping the wires connecting electrode R and C the measured immittance should not change. This is guaranteed by the reciprocal theorem. If it does change then one of the electrodes might be too small and be in the non-linear region. This is an important test to ensure that the polarization impedance of electrode C and impedance of R does not contribute to the measurement.

If the metals of the R and M electrode is not the same, a DC potential will be generated. This is rarely wanted and care should be taken ensuring they are of the same material, especially when the same type of electrode cannot be used at both sites. This problem also arises if the tissue connected to generates an endogenic DC potential. The material measured on will then be under constant polarization as the operational amplifier will compensate for the potential difference trying to drive the inputs to zero volt difference.

Combining the three electrode setup with making the relative size of the measurement electrode much smaller than the reference electrode increases the effect of the monopolar measurement.

### 2.4.6 Electrode size and geometry

Both noise and what is measured is affected by the size and geometry of the electrode. Two common skin surface electrode designs are shown in figure 2.11. Here the metal-electrolyte interface area is called electrode area (EA) and the electrolyte-skin interface area is called the effective electrode area (EEA). EA determines the polarization impedance and EEA determines the skin impedance. Large EA implies low electrode polarization impedance. Large EEA on a measuring electrode implies an averaging effect and loss of spatial resolution. Large EEA on a stimulating electrode implies higher excitable tissue volume. (Grimnes and Martinsen, 2006, 2008)

### 2.4.7 Electrode noise

According to (Grimnes and Martinsen, 2008, p. 264) there are three rules that are important regarding electrode noise:

1. Less noise the larger the electrode area, because of the averaging effect.

2. The more the electrode is polarizable, the more noise it will generate.

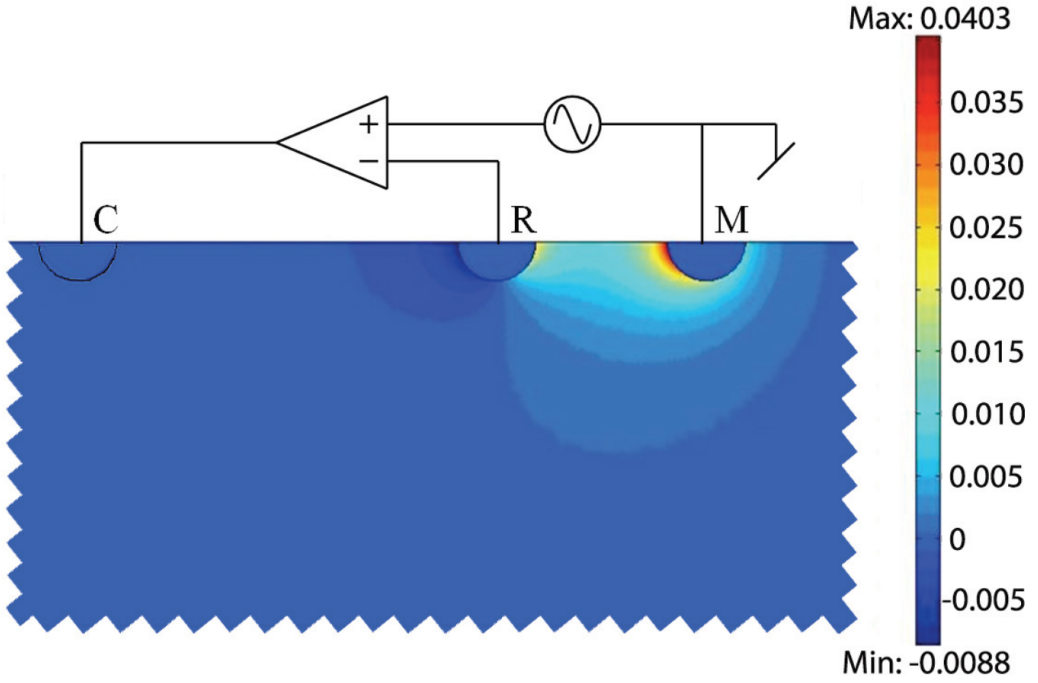



Figure 2.10: Shows the sensitivity area for a three-electrode setup in an infinite homogenous material. Taken from (Kalvoy, 2010)

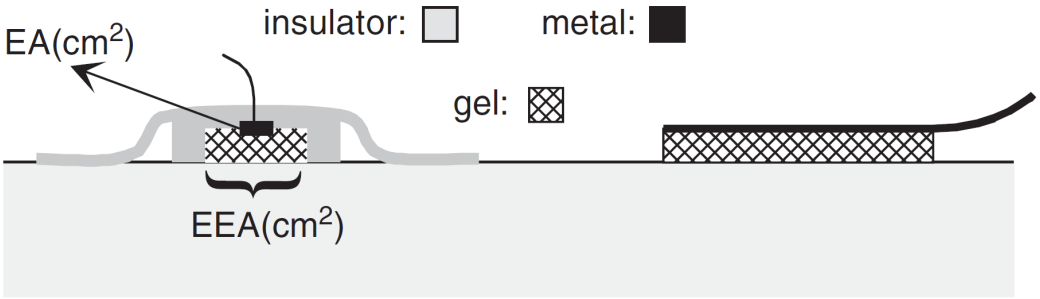



Figure 2.11: Two common skin surface electrode designs. EA is the electrode area and EEA is the effective electrode area. Taken from (Grimnes and Martinsen, 2006)

3. A more diluted contact electrolyte will generate more noise.

In addition to this, sudden spikes of millisecond duration and hundreds of microvolt amplitude can be generated. Also a non-uniform electrode surface will generate noise.

### 2.4.8 Needle electrode

One of many invasive electrodes is the needle electrode. When this electrode is used in a monopolar setup where the other electrodes are relatively much larger, only the material's impedance very close to the needle contribute to the measured impedance. A FEM calculation of this have been tried by (Hoyum et al., 2010). An insulated needle was placed in a saline solution and the measurement frequency was 100 kHz. The insulated needle had an active electrode area of 0.3 mm2 and the thickness of the insulation was about 26 $\mu m$. More about the boundary conditions and material constants can be found in (Hoyum et al., 2010) and more about general impedance of needle electrodes can be found in (Kalvoy et al., 2010). Figure 2.12 shows the sensitivity field from the simulation. A 97% sensitivity radius at 3.75 mm and 77.3% sensitivity radius at 1 mm. This shows that the part of the material very close to the needle electrode largely determines the immittance measured. This effect can be utilized to do for example tissue discrimination. More about tissue discrimination using needle electrode can be found in (Kalvoy et al., 2009).

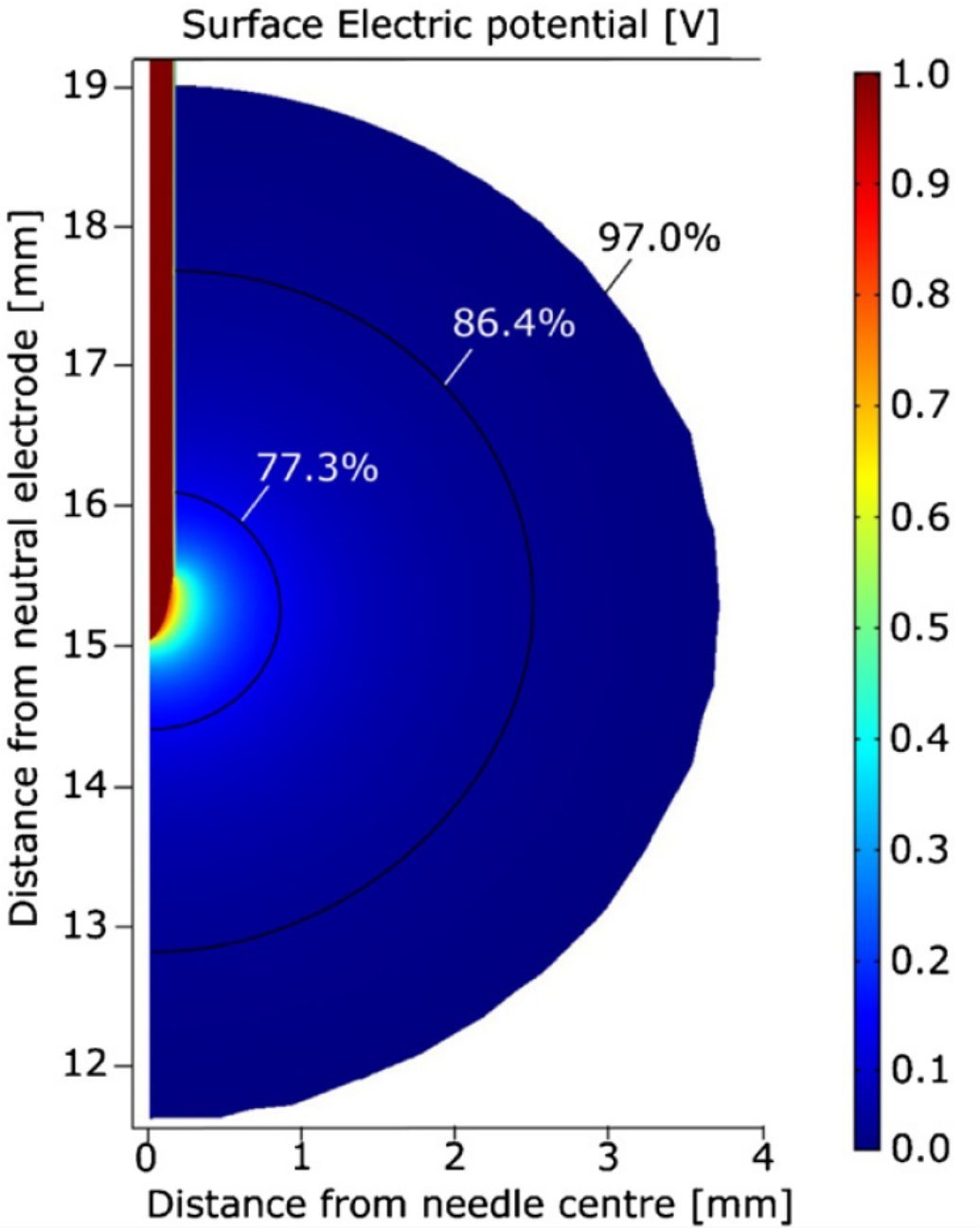



Figure 2.12: Electrical potential distribution and sensitivity field of a needle electrode in a saline solution. The scale on the right shows the voltage factor, where 1.0 is the maximum voltage which can be found on the needle electrode. Taken from (Kalvoy, 2010)

# Chapter 3

# Electronics Theory

## 3.1 Sine wave

The sine wave is often seen in AC applications. In this section the relationship between different parameters describing the sine wave will be examined.

The peak value of a sine wave is is often labelled $V_p$ or $V_{peak}$ and is the top/bottom value of the sine wave from the center line. The average value is calculated by only taking the average of a half cycle as the average of a full cycle would be zero. The relationship between the average and the peak is:

$$V_{avg} = V_p \cdot 0.637 \tag{3.1}$$

The connection between the peak value and the root mean square (RMS) is:

$$V_p = \sqrt{2} \cdot V_{RMS} \tag{3.2}$$

If an AC current or voltage has a RMS value of 5 volts or ampere respectively, this is equal to the power in a DC current or voltage with the same current or voltage.

Equation 3.1 and 3.2 shows that the relation between the RMS and average can be written as:

$$V_{RMS} = V_{avg} \cdot 1.11 \tag{3.3}$$

## 3.2 Direct Digital Synthesizer

The DDS outputs a sinusoidal signal with a frequency that is set by a digital numeric control. The DDS requires an external timing reference, e.g. a

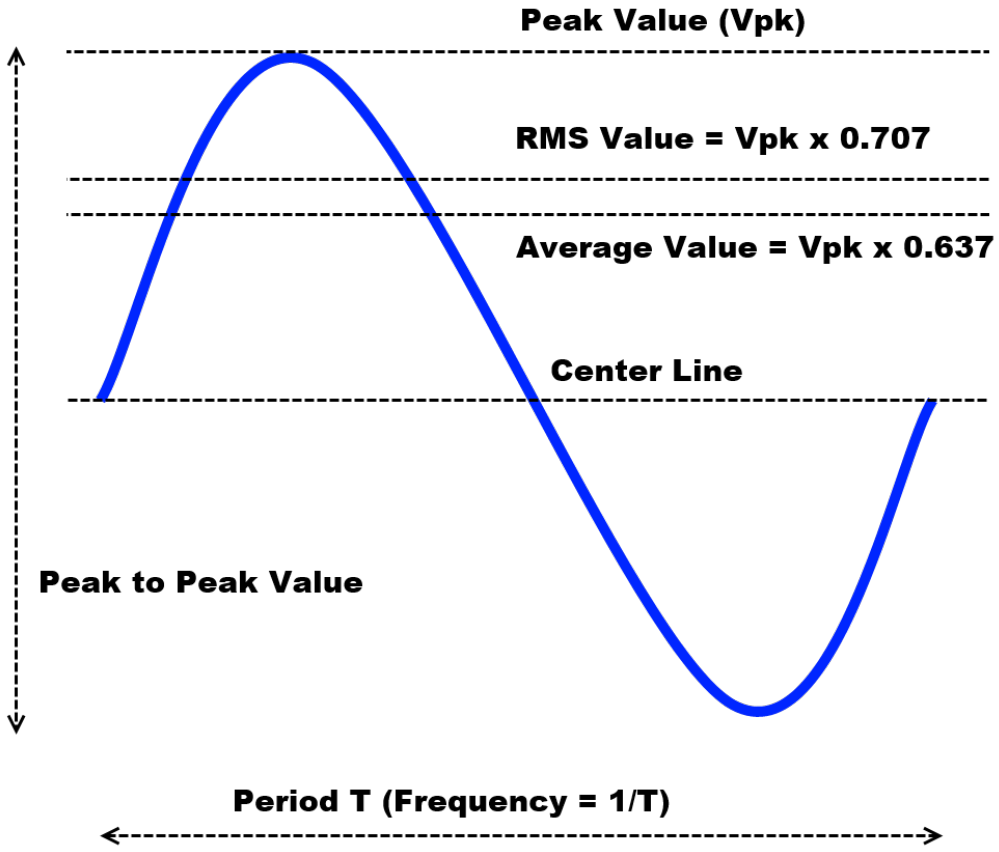



Figure 3.1: Sine wave with the most commen sine wave characteristics.

crystal oscillator. This clock will be the sampling clock of the DDS. The frequency output range of a DDS is normally from DC to half of the external clock frequency.

A DDS consists of three building blocks. The accumulator, sine lookup table (angle-to-amplitude converter) and a digital-to-analog converter. This can be seen in figure 3.2.

The accumulator increments a number on each clock pulse resulting in a ramp as shown in figure 3.2. The accumulator is often 12 to 24 bits. When the accumulator overflows, one period of a sinusoidal signal is achieved and the accumulator starts incrementing from the reminder value, if any. This means that the larger the increment is, the faster it will overflow and higher the output frequency. The size of the increment is one of the inputs to a DDS and is often called the frequency (tuning) word.

The value output by the accumulator always corresponds to a phase between 0 and 360 degrees. The sine lookup table simply converts the phase angle to a corresponding amplitude by looking up the phase angle in a read-only memory (ROM) and output the amplitude. This creates a discrete sinusoidal signal on the angle-to-amplitude converters output. This can be seen in figure 3.2.

The digital-to-analog converter (DAC) converts the discrete digital sinusoidal signal to a continuous analog sinusoidal voltage or current signal as seen in figure 3.2.

Even though usually not a part of the DDS, a low pass filtering circuit is added after the DDS output to smooth the analog sinusoidal waveform and remove unwanted harmonics/images.

The output frequency $f_{out}$ of the DDS is:

$$f_{out} = \frac{f_{clk} F_w}{2^N} \tag{3.4}$$

where $f_{clk}$ is the external clock frequency, $F_w$ the frequency word and $N$ the number of accumulator bits.

## 3.3 Current Measurements

Two basic techniques used for measuring small currents are the shunt ammeter and the transimpedance amplifier (Keithley Instruments, 2004, c. 1.5.2). The transimpedance amplifier is also known as a feedback ammeter.

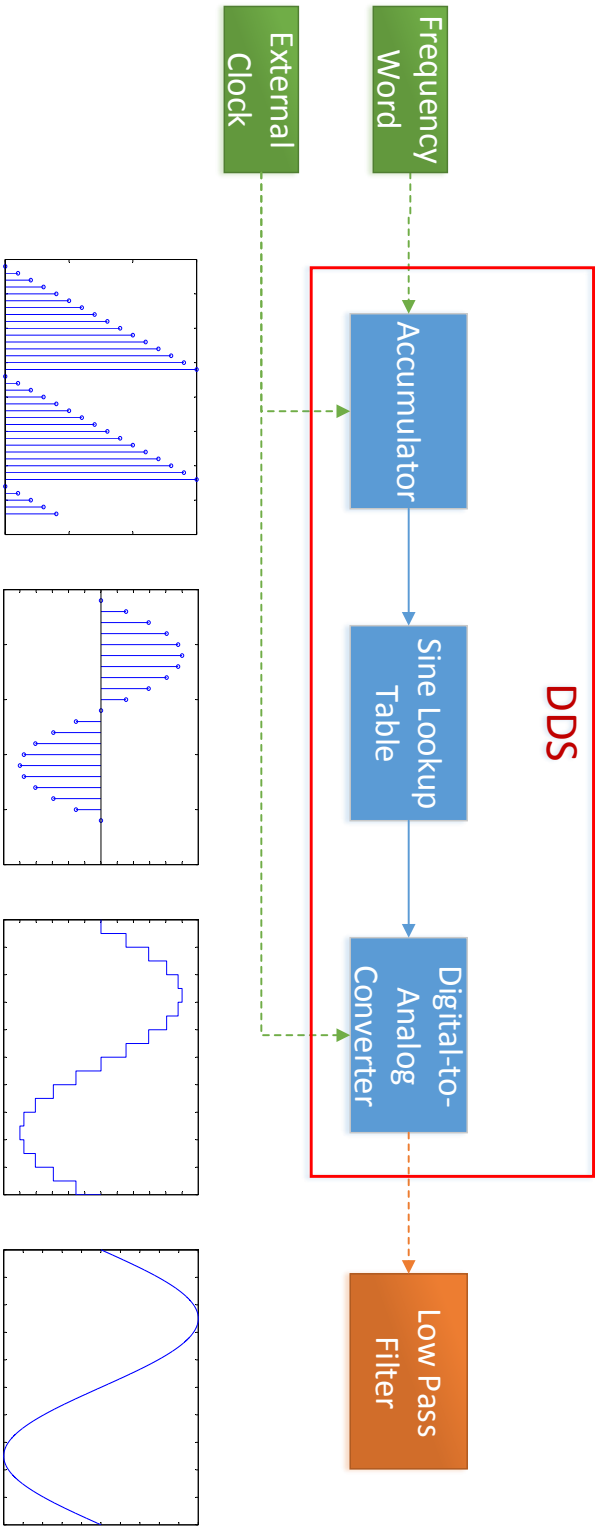



Figure 3.2: The building blocks of a DDS

### 3.3.1 Shunt Ammeter

A shunt ammeter can be made by placing a shunt resistor on the positive input of a voltage follower. The basic circuit will look like in figure 3.3.

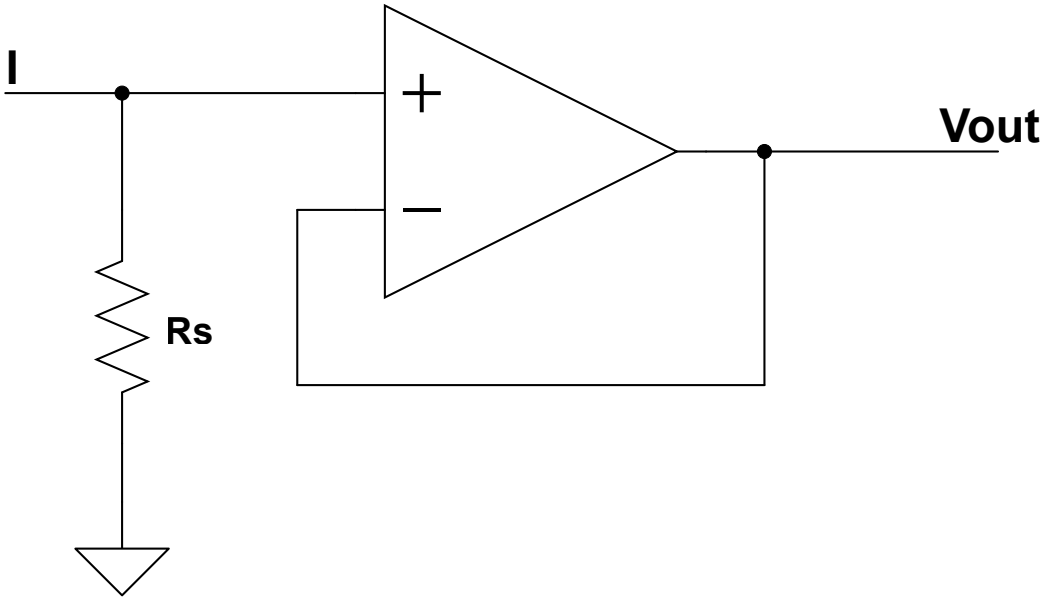



Figure 3.3: Basic shunt ammeter

Because of the high input impedance of the operational amplifier, the input current $I$ flows through the shunt resistor $R_s$ creating a potential on the positive input. The op-amp will try to keep the same potential on both inputs and the output voltage $V_{out}$ is then:

$$V_{out} = IR_s \tag{3.5}$$

The shunt resistor value should be made just big enough to ensure that the wanted voltage output is received. Some of the advantages of a lower shunt resistor value is better accuracy, time and temperature stability and lower input time constant, but lower signal-to-noise (SNR) ratio (Keithley Instruments, 2004, c. 1.5.2).

### 3.3.2 Transimpedance Amplifier

The transimpedance amplifier is a inverting negative feedback operational amplifier where the current signal is provided on the negative input. This is shown in figure 3.4.

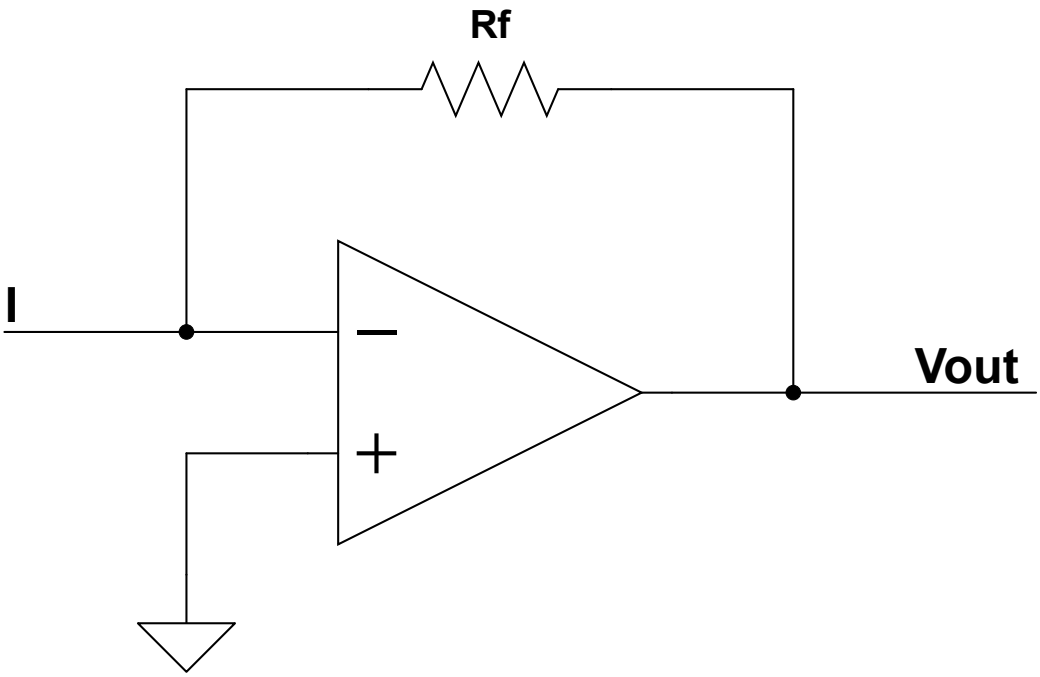



Figure 3.4: Basic transimpedance amplifier.

In this circuit the input current $I$ flows through the feedback resistor $R_f$. If the op-amp has a relative low input offset current the output voltage $V_{out}$ is:

$$V_{out} = -IR_f \tag{3.6}$$

In this circuit, the size of the output signal is directly proportional to the feedback resistor $R_f$. The voltage burden is low and the high gain bandwidth product (GBP) of the operational amplifier ensures higher frequency operation than the shunt ammeter.

### Feedback compensation

To increase the stability, bandwidth and reduce ringing on the output signal of a transimpedance amplifier, feedback compensation is used. Feedback compensation is done using a feedback capacitor $C_f$ in parallel with the feedback resistor $R_f$, as shown in figure 3.5.

To ensure stability, a capacitor value that provides a phase margin of at least 45 degrees should be calculated. TI (2013). Phase margin is 180 degrees subtracted from the difference between the input and output phase

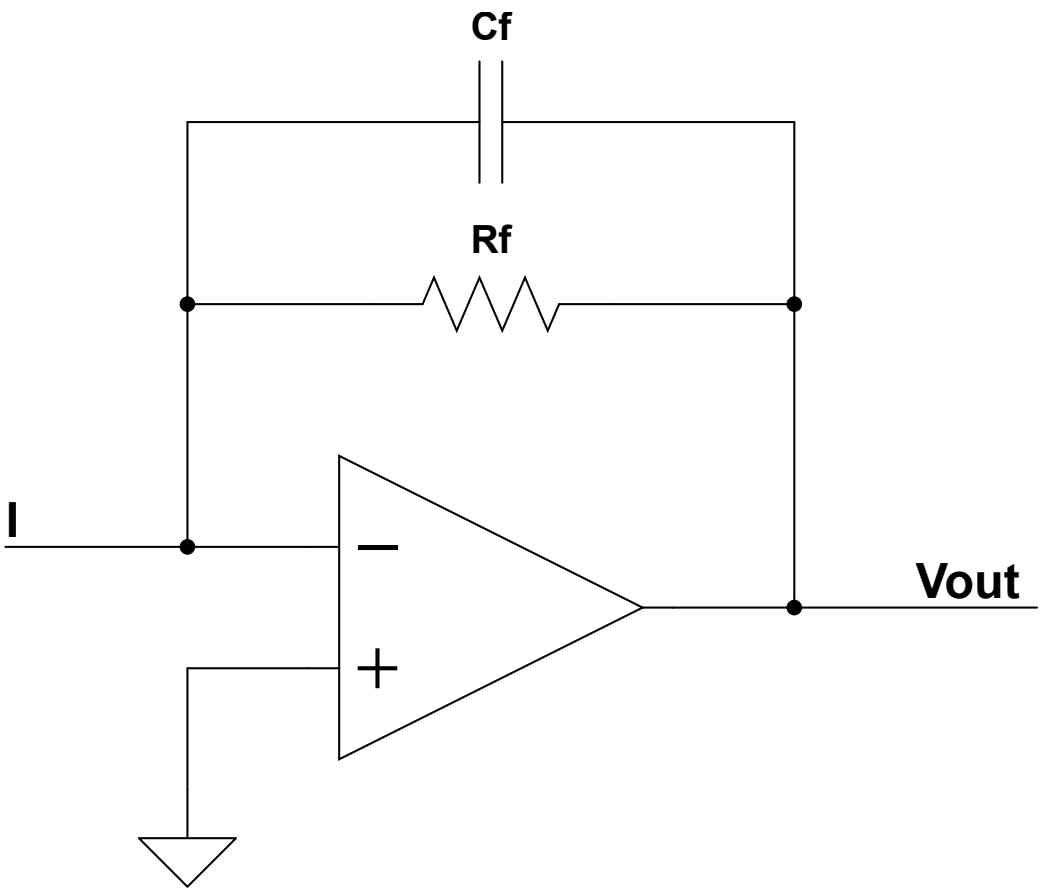



Figure 3.5: Transimpedance amplifier with compensation.

of an amplifier. The value of the feedback capacitor $C_f$ with a 45 degrees phase margin is:

$$C_f = \sqrt{\frac{C_s + Ci}{2\pi R_f f_{GBP}}} \tag{3.7}$$

where $C_s$ and $C_i$ is the shunt and input capacitance and $f_{GBP}$ the gain bandwidth product.

The cut-off frequency calculation for a transimpedance amplifier is not done the same way as with an inverting amplifier. If it were, using a high value feedback resistor like a 1 $M\Omega$ with an op-amps with large GBP would still only give a bandwidth in the thousands of hertz.

The transimpedance amplifiers cut-off frequency (-3dB), if the feedback capacitor is calculated using equation 3.7, is:

$$f_{-3dB} = \sqrt{\frac{f_{GBP}}{2\pi R_f C_f}} \tag{3.8}$$

It is recommended by Bhat (2012) to overcompensate to provide sufficient guardband to account for the up to ± 40 % variations in an op-amp's bandwidth and feedback capacitors tolerance.

## 3.4 Virtual Ground

Virtual ground is a steady reference potential that is different from ground. The simplest virtual ground reference can be made using a voltage divider as shown in figure 3.6.

Here the output voltage $V_{out}$ is:

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \tag{3.9}$$

where $V_{in}$ is the input voltage and $R_1$ and $R_2$ the resistor values. A voltage follower can be cascaded after the voltage divider to assure lower output impedance and more stable output reference voltage.

## 3.5 Decoupling capacitors

Decoupling capacitors (also called bypass capacitors) are used to reduce the effect noise has on a circuit element. A decoupling capacitor is connected as a shunt as close as possible to the circuit elements power supply pin. Most

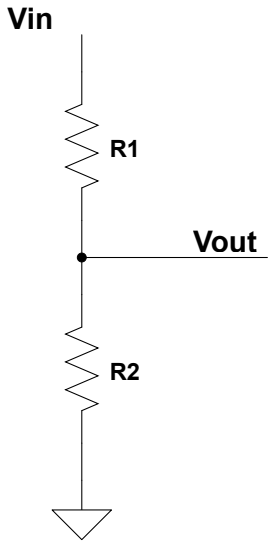



Figure 3.6: A voltage divider.

circuit elements ideally want a constant DC voltage level, but external noise from the environment and other circuit elements are superimposed on the DC voltage. The decoupling capacitor provides a low impedance path for these transients and works as a local energy storage for the circuit element. When there is a drop in voltage, the capacitor compensate by releasing energy and when there is a spike it charges.

The value of the capacitance used depends on the frequencies one want to suppress. A larger capacitance value is better at suppressing lower frequencies and vice versa. Decoupling capacitors are widely used in most electronics designs.

## 3.6 Comparator

A comparator is a specialized op-amp circuit that compares two voltage or current signals and the binary output signal shows which is larger. A comparator is shown in figure 3.7.

If the positive input $V_{in+}$ is larger than the negative input $V_{in-}$ then the output $V_{out}$ is usually close to the value of the comparators power supply. If the negative input is larger than the positive input, then the output is usually ground. Because the output only has two states, it can be suitable to connect directly to a digital circuit.

The output of the ideal comparator changes its state when the difference of the inputs are zero. Due to noise at the inputs, unwanted very fast changes

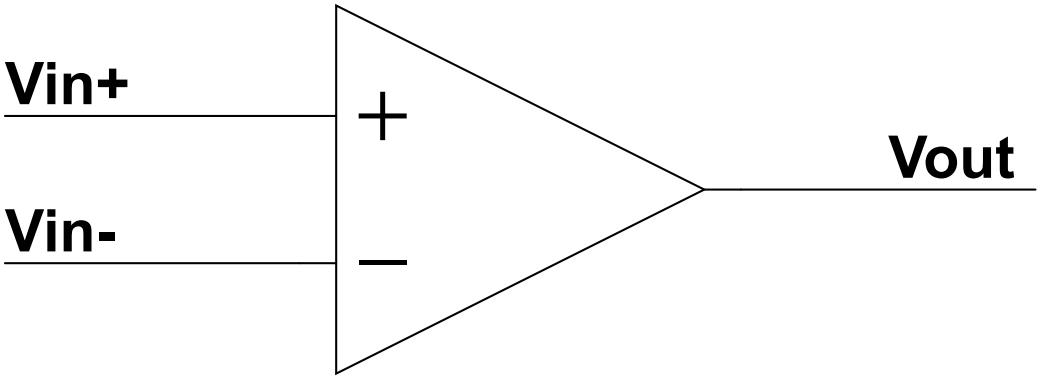



Figure 3.7: A comparator.

between the two output states can occur when the difference between the inputs is close to zero. To avoid this behavior, internal hysteresis is often integrated or must be provided externally by adding a resistor to the feedback loop from the output to the positive input.

Speed and power consumption can be important parameters and in general, the faster the comparator, the more power it consumes.

One of the many uses of a comparator is as a zero crossing detector (ZCD). This is used to detect when the polarity of an AC signal changes. The output of the comparator will be high when the polarity is positive and low when the polarity is negative.

## 3.7 AC coupling

AC coupling is used to remove the DC signal. This is often done by placing a capacitor in series with the signal one want to remove the DC signal from. This means that only the AC signal may pass through to the next part of the circuit. This is also called capacitive coupling and the capacitor is often called a DC blocking capacitor or decoupling capacitor. The decoupling capacitor in combination with the input impedance of the next stage in the circuit forms a high pass filter. This high pass filter attenuate lower frequencies so it is important to know the frequency characteristics, as the cut-off frequency. It is possible to accurately control the cut-off frequency as shown in figure 3.8.

Here the shunt resistor and the DC blocking capacitor forms the high pass filter. The cut-off frequency $f_{-3dB}$ is then:

$$f_{-3dB} = \frac{1}{2\pi RC} \tag{3.10}$$

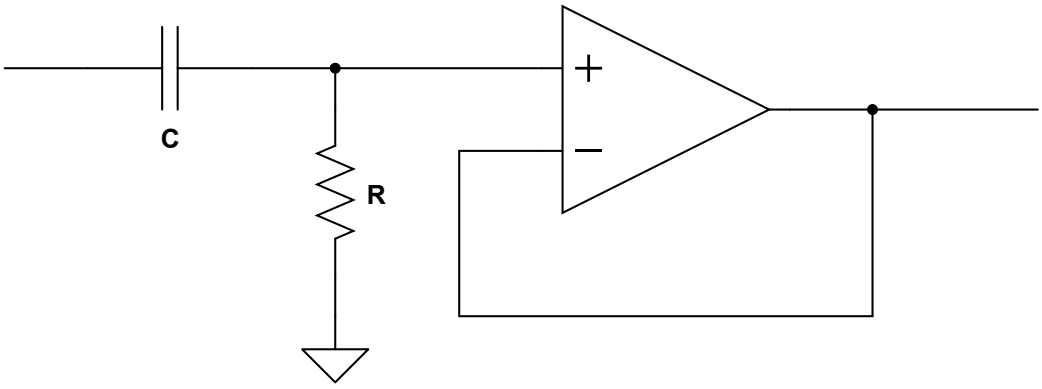



Figure 3.8: A non-inverting AC coupling circuit.

## 3.8 Analog-to-Digital Converter

An ADC converts (or samples) a time-continuous physical quantity to a discrete time-and amplitude digital signal. Often this physical quantity is a voltage. The sampled digital signal has a finite number of bits that can represent the amplitude of the signal. This is called the amplitudes resolution and equals the least significant bit (LSB) and for a full-scale voltage range V the resolution of an N-bit ADC is:

$$Resolution = LSB = \frac{V}{2^N - 1} \tag{3.11}$$

The sampling rate is the time resolution and must be set correctly for a given setup to be able conduct a successful measurement.

## 3.9 Operational Amplifier

An operational amplifier (op-amp) is used to perform mathematical operations on analog signals. A standard op-amp has a differential input, one output and two power supply connections. The standard op-amp symbol and its connections are shown in fig 3.9.

Some of the operations that can be done is adding, subtraction, multiplication, division, integration and derivation. An ideal op-amp has infinite input impedance, zero output impedance and infinite open loop gain. A real op-amp does not have this and its parameters are controlled by external components. Some of these op-amp configurations are shown in the following subsections.

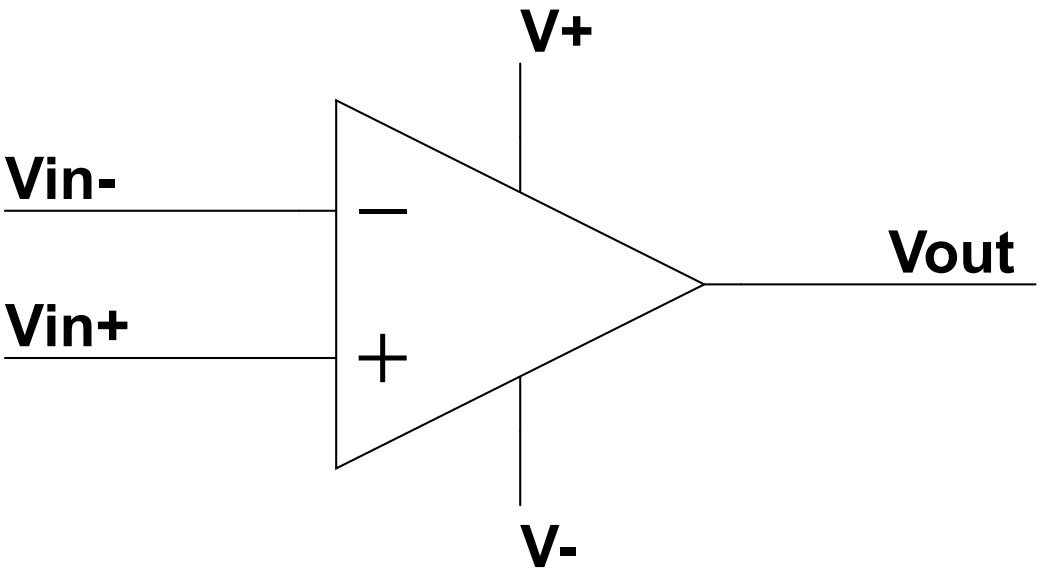



Figure 3.9: An operational amplifier.

### 3.9.1 Voltage Follower

A voltage follower (also called a voltage buffer or just buffer) is shown in figure 3.10.

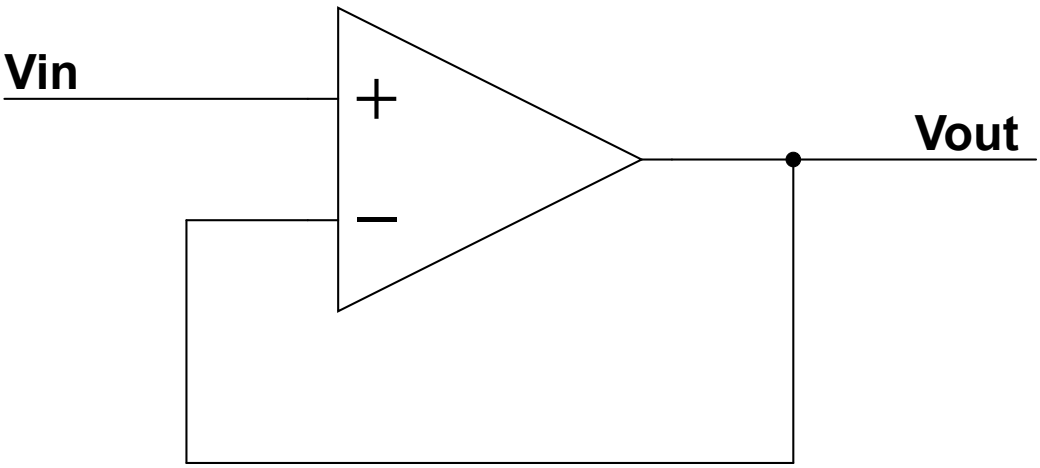



Figure 3.10: A voltage follower.

It simply ensures that the output voltage is the same as the input voltage:

$$V_{out} = V_{in} \tag{3.12}$$

but it also has some other properties.

The input impedance of this circuit is determined by the op-amps input impedance and is very high. This means that the voltage follower does not load the source, because the current draw on the op-amps input is negligible.

The output impedance of an op-amp is very low. For the voltage follower this means that the op-amp is able to act as an ideal voltage source. An ideal voltage source can maintain a voltage independent of the load or the output current. This of course only applies within the limits given in the op-amps data sheet.

### 3.9.2 Negative Feedback Amplification

There are two main types of negative feedback amplifiers. The non-inverting and the inverting amplifier. They can both amplify the input signal on their outputs, but have their advantages and disadvantages.

#### Non-inverting Amplifier

The non-inverting amplifiers output voltage changes with the same phase as the input voltage. A typical non-inverting amplifier is shown in figure 3.11.

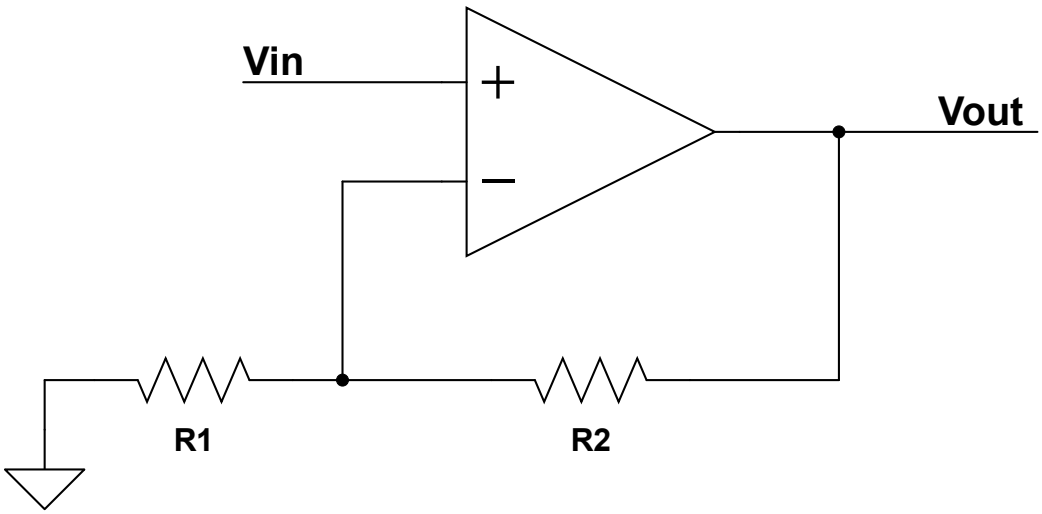



Figure 3.11: A non-inverting operational amplifier.

The amplification is usually called gain ($A$) and if the open-loop gain ($A_{OL}$) is very large, then the gain can approximately be calculated as:

$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_2}{R_1} \tag{3.13}$$

### Inverting Amplifier

The inverting amplifiers output voltage changes 180 degrees out of phase with the input voltage. A typical inverting amplifier is shown in figure 3.12.

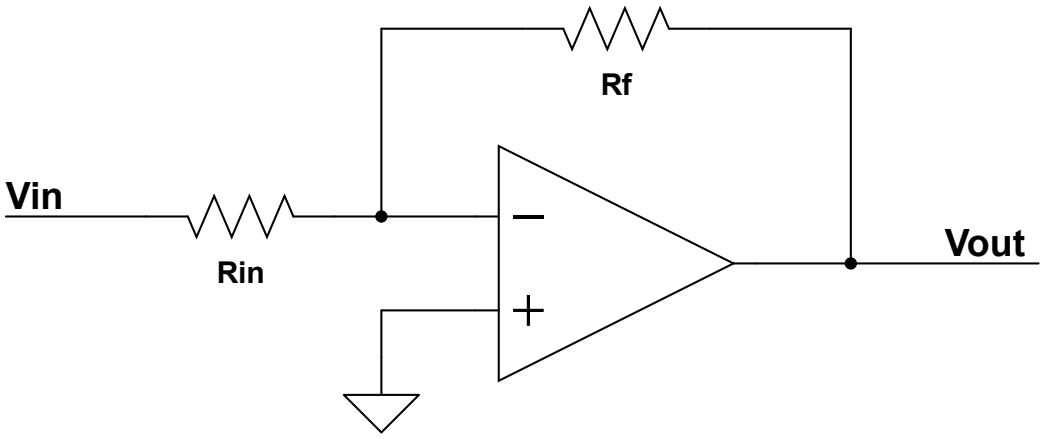



Figure 3.12: An inverting operational amplifier.

If the open-loop gain is very large the gain can approximately be calculated as:

$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}} \tag{3.14}$$

### Advantages and Disadvantages

The input impedance in a non-inverting amplifier is very high and is determined by the op-amps input impedance. For an inverting amplifier the input impedance is $R_{in}$ as the point where the feedback loop meets the negative input has zero potential and therefor is virtually ground.

The inverting input can also act as an attenuator by making $R_f < R_{in}$, while the gain of the non-inverting op-amp approaches 1 when $R_1 \to \infty$.

## 3.10 Single Supply

Dual supply systems are usually powered by two supplies equal in magnitude, but with reverse polarity. For a voltage powered op-amp the center voltage between the supplies is connected to ground. This makes it possible for any input source connected to ground to be referenced to the center voltage of the supplies and the output voltage is then automatically referenced to ground.

The input and output voltage can normally have values within the power supplies voltage range, both negative and positive.

Single supply systems are powered by a single supply and does not have the same ground reference as a dual supply system has. When an input source is connected to ground, it is not referenced to the single supply's center, but the lower power supply rail, which is ground. This is equivalent to the negative supply in a dual supply system. This means a single supply op-amp is not able to handle negative input signals for a non-inverting amplifier or positive input signals for an inverting amplifier, as it would force the output to go negative, which it cannot. It is therefore necessary to bias the single supply op-amp to avoid this problem. An example of this can be seen in figure 3.13.

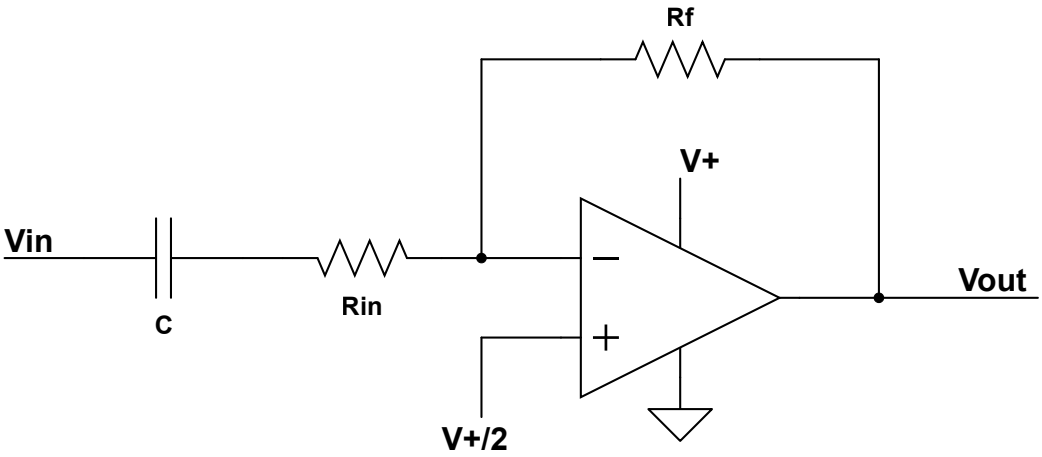



Figure 3.13: A biased AC coupled inverting operational amplifier.

Here the op-amp is biased with half of the op-amps power supply on the positive input. The capacitor on the input assures that only AC voltage is passed through from the input source and vice versa. This circuit superimposes the passing AC voltage on the bias voltage at the op-amps output, even if the input source signal has positive polarity. The output can only be positive so the input signal amplitude can maximally be half of the single supply voltage if it is a sine wave centered around zero volt. This also requires that the bias voltage is half the single supply voltage.

As demonstrated, it is possible to use single supply systems to perform the same operations as dual supply systems with some modifications.

The motivation behind using single supply systems is the lower power consumption demands of portable battery-powered devices. Also only one battery is needed.

## 3.11 Noise in Analog circuits

Noise is present in all analog circuits and three of the most common sources of noise is presented in this section. Noise affects a systems performance and one of the most used parameters to determine the relationship between the signal and noise is the signal-to-noise ratio (SNR). Steps can be taken to reduce noise when the noise source and what type of noise it present in part of a system.

### 3.11.1 Thermal Noise

The random movement of charge carriers generates thermal noise (also called Johnson noise among other names). Thermal noise is present whether a voltage is applied or not and the more heated the conductor, the more thermal noise will be present. Only at absolute zero degrees is no thermal noise present.

Thermal noise is considered white noise, which means it has a uniform power density in the frequency spectrum.

At frequencies below 100 MHz, thermal noise can be calculated using Nyquist's relation (TexasInstruments, 2002, c. 10.8):

$$E_{th} = \sqrt{4kTRB} \tag{3.15}$$

or

$$I_{th} = \frac{E_{th}}{R} = \sqrt{\frac{4kTB}{R}} \tag{3.16}$$

where $E_{th}$ is the thermal noise voltage in Volts rms, $I_{th}$ is the thermal noise current in Amps rms, $k$ is the Boltzmann's constant, $T$ the temperature in Kelvin, $R$ the resistance in Ohms and $B$ the bandwidth in Hertz.

### 3.11.2 Shot Noise (Schottky Noise)

Charge carriers are a flow of discrete charges and each time an electron passes a potential barrier noise is generated. This because potential energy is built up until there are enough energy to pass the potential barrier and then released in order for the electrons to pass.

Shot noise is associated with current flowing and there are no shot noise present when there are no current flowing.

Shot noise is also independent of temperature and has a uniform power density in the frequency spectrum.

Shot noise is mainly present in semiconductors, but also in any conductor due to imperfections and impurities in the metal.

If the semiconductor PN-junction is forward biased the rms shot noise current is:

$$I_{sh} = \sqrt{2qI_{dc}B} \tag{3.17}$$

where $q$ is the electron charge, $I_{dc}$ is the average forward DC current in Amperes and $B$ the bandwidth in Hertz.

and the rms shot noise voltage is:

$$E_{sh} = kT\sqrt{\frac{2B}{qI_{dc}}} \tag{3.18}$$

where $k$ is the Boltzmann's constant and $T$ the temperature in Kelvin.

Now we can see that the shot noise is inversely proportional to the current. This means that in most conductors shot noise will be very small.

### 3.11.3 Flicker Noise

Flicker noise (also called 1/f noise) is present in all active and many passive devices. Flicker noise increase with decreasing frequency, hence the name 1/f. Flicker noise is associated with DC current and has the same power content in each decade (TexasInstruments, 2002, c. 10.8).

The shot noise voltage is:

$$E_f = K_V\sqrt{\ln\left(\frac{f_{max}}{f_{min}}\right)} \tag{3.19}$$

where $K_V$ is a proportionality constant in volts representing $E_f$ at 1 Hz, $f_{max}$ and $f_{min}$ are maximum and minimum frequencies in Hertz.

The shot noise current is:

$$I_f = K_I\sqrt{\ln\left(\frac{f_{max}}{f_{min}}\right)} \tag{3.20}$$

where $K_I$ is a proportionality constant in amperes representing $I_f$ at 1 Hz.

## 3.12 Transimpedance Amplifier Noise Analysis

There are three main sources of noise in a transimpedance amplifier. The op-amps input voltage noise, its input current noise and the feedback resistors thermal noise. This section is based on the article by Orozco (2013).

Knowing the transimpedance amplifier's cut-off frequency from equation 3.8 the equivalent noise bandwidth (ENBW) can be calculated:

$$\text{ENBW} = f_{-3dB} \cdot \frac{\pi}{2} \tag{3.21}$$

All noise calculations in this section is root mean square (RMS) values.

The feedback resistor's noise will appear directly on the output as:

$$N_{Rf} = \sqrt{4kT \cdot \text{ENBW} \cdot R_f} \tag{3.22}$$

where $k$ is the Boltzmann constant and $T$ the temperature in Kelvin.

The op-amp's current noise will appear on the output after going through the feedback resistor as:

$$N_{current} = I_n R_f \cdot \sqrt{\text{ENBW}} \tag{3.23}$$

where $I_n$ is the current noise density of the op-amp.

With the assumptions that the first pole and zero of the output noise density is minimum a decade lower than the second pole and that the output noise is equal to the plateau noise that is:

$$N_2 = e_n\left(\frac{C_f + C_{sh} + C_i}{C_f}\right) \tag{3.24}$$

where $e_n$ is the voltage noise density, $C_f$ the feedback capacitor, $C_{sh}$ the shunt capacitance and $C_i$ the op-amps input capacitance.

An estimation of the voltage noise is then:

$$N_{voltage} = N_2 \cdot \sqrt{\frac{\pi}{2} f_{p2}} \tag{3.25}$$

where $f_{p2}$ is the second pole and defined as:

$$f_{p2} = f_{GBP} \cdot \frac{C_f}{C_f + C_{sh} + C_i} \tag{3.26}$$

where $f_{GBP}$ is the gain bandwidth product.

The three noise sources are independent and Gaussian meaning that the total noise is the root-sum-square (RSS):

$$N_{total} = \sqrt{N_{Rf}^2 + N_{current}^2 + N_{voltage}^2} \tag{3.27}$$

A low pass filter on the transimpedance output can greatly reduce the total noise if $f_{p2}$ is much higher than the signal bandwidth.

# Chapter 4

# Digital Theory

## 4.1 ARM

### 4.1.1 What is ARM

ARM is a family of microprocessors and microcontrollers based on the Reduced Instruction Set Computer (RISC) architecture. The RISC architecture is often associated with low cost, less heat and low power computer processors. ARM Holdings, which own the ARM family, licenses its chip design and instruction set to third-party companies. Third party companies then add, e.g., memory, peripherals, wireless radios etc. This is what for example ST Microelectronics and Texas Instruments do.

### 4.1.2 What is ARM Coretex-M3

The ARM Cortex-M3 is one of ARM's microcontroller cores based on the ARMv7M architecture which is a Harvard architecture with separate code and data busses. It is a 32-bit microcontroller with a mixed 16- and 32-bit instruction set called Thumb2. It is bi-endian, but little endian by default. It supports one clock-cycle hardware multiplication and division and the multiply-and-accumulate (MAC) instruction. The Cortex-M3 is a high-performance low-cost microcontroller widely used in the industry.

### 4.1.3 Nested Vector Interrupt Controller (NVIC)

Cortex-M3 have an advanced interrupt controller called NVIC. Cortex-M3 can support up to 256 different priority levels and supports both level and edge interrupts. A higher number corresponds to a lower priority.(ARM, 2010, p. 129) The STM32 only utilize 4 bits for priority levels, which means

there are 16 levels. NVIC supports nesting of interrupts, which means interrupts with higher priority can pre-empt lower priority interrupts when they are being serviced. NVIC also supports tail-chaining interrupts. This means that if there is a pending interrupt when already servicing another interrupt, the microcontroller skips context switching and the next interrupt is serviced only 6 cycle after the previous one ends. (ARM, 2006, p. 108)

### 4.1.4 Cortex Microcontroller Software Interface Standard (CMSIS)

The ARM microcontrollers are complex and writing macros to all peripheral memory registers, bitmasks and interrupt vectors is work that would have to be repeated by every developer. To prevent this ARM has released a portable and vendor-independent hardware abstraction layer called CMSIS. CMSIS works on all Cortex based microcontrollers and provide a small, but important abstraction from hardware. This will make it easier start using the Cortex family, save development time and increase standardization. CMSIS also contains function calls to core peripherals like the NVIC and SysTick timer.

### 4.1.5 CMSIS Digital Signal Processing Library

CMSIS also contains a digital signal processing (DSP) library. This library adds support for complex number, PID regulation, filtering, matrix, statistics and fast fourier transform (FFT). It has support for both single precision and fixed-point numbers. The code is written as a mix of C and assembly code optimized for the ARM Cortex instruction set.

## 4.2 STM32

STM32 is a microcontroller based on the ARM Cortex series. STM32F0xx is based on Cortex-M0, STM32F1xx and STM32F2xx on Cortex-M3 and STM32F3xx and STM32F4xx on Cortex-M4. There are also other STM32 microcontrollers based on ARM Cortex that specializes in low power, connectivity or touch sensing. The microcontroller chosen in this thesis is the STM32F103VC. This microcontroller is based on the ARM Cortex-M3 core and has many peripherals added by STMicroelectronics.

### 4.2.1 Clock tree

STM32 has several internal clocks and support different external clocks. The clock tree is shown in the figure 4.1 and should give an overview of which and how the different clock signals are logical connected in the microcontroller.

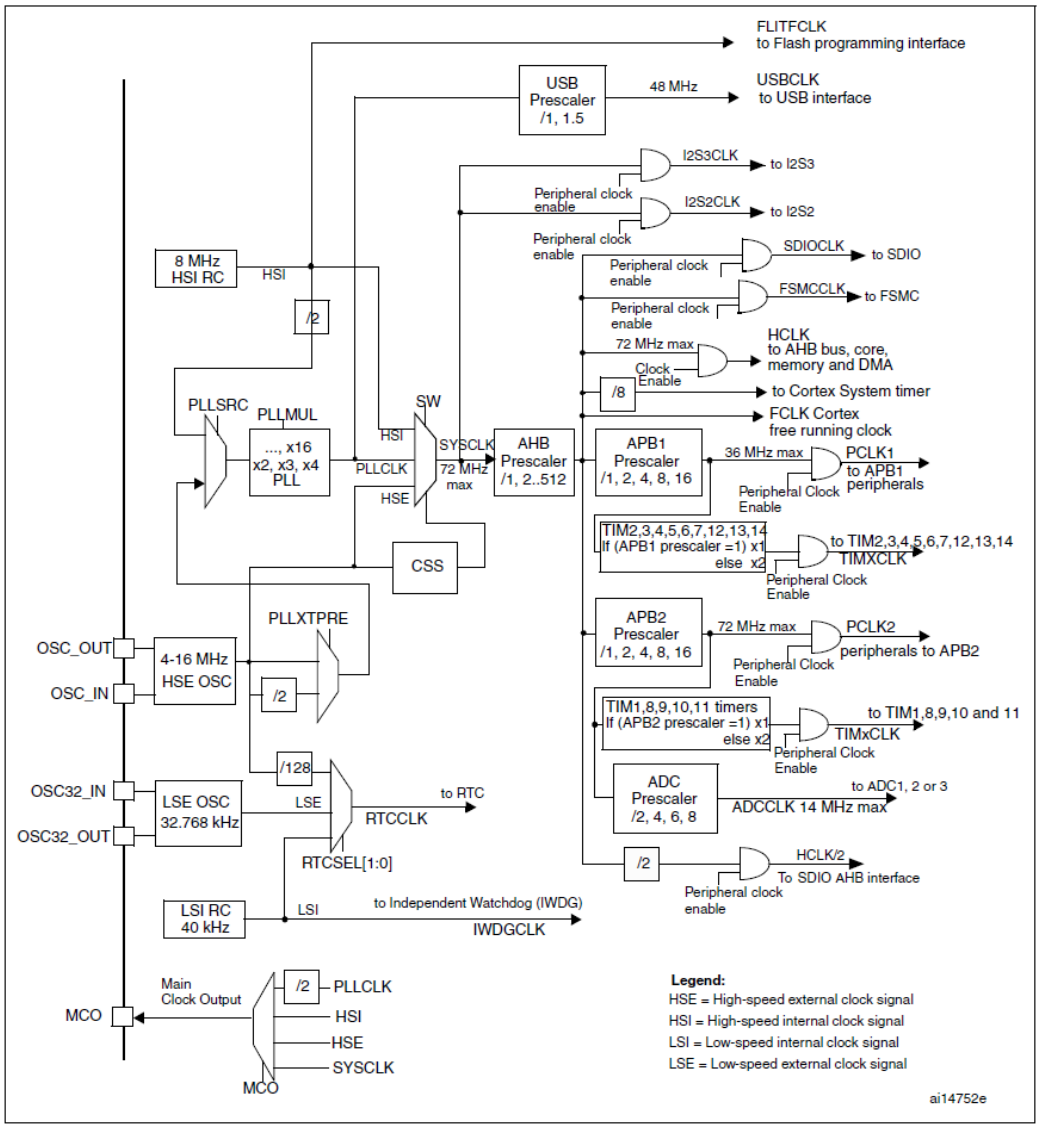



Figure 4.1: STM32 clock tree. Taken from (STM32, 2011a, p. 90)

In this thesis, the external high speed (HSE) clock is used as the system clock and provides most of the STM32 peripherals through the ARM Advanced High Speed Bus (AHB) and the two ARM Advanced Peripheral

Busses (APB). The Real Time Clock (RTC) is connected to an external high precision 32.768 kHz oscillator and the Independent Watchdog (IWDG) to an internal inaccurate 40 kHz (±40%) RC oscillator.

### 4.2.2 System clock

After reset the STM32 will start up using the high speed internal (HSI) oscillator. If an external high speed oscillator is available, it must be turned on. After it has stabilized, it is fed to the Phase-Locked Loop (PLL) and when the PLL has stabilized, it is selected as the system clock. It is possible to use the HSI oscillator to run the STM32 at up to 64 MHz (STM32, 2011a, p. 90), but it is an inaccurate and unstable 8 MHz RC clock source. The HSI is unsuitable for serial communication and timer peripherals. (Martin, 2009, p. 41)

### 4.2.3 Watchdog

The STM32 independent watchdog timer is a 12-bit count down timer that will reset the microcontroller when it reaches zero. (STM32, 2011a, p. 475) It is therefore necessary to update the watchdog before timeout unless a reset is wanted. This means a system reset will happen if the software blocks or delay too long before the watchdog timer is updated.

### 4.2.4 Direct Memory Access (DMA)

DMA makes it possible to offload the Cortex CPU when it comes to data transfers between memory-to-memory, memory to peripheral, peripheral to memory or peripheral-to-peripheral. This means the CPU can do other operations while data is moved. The STM32 have two DMA controllers with seven channels on DMA1 and five channels on DMA2. Each channel is connected to different peripherals in hardware. DMA has four levels of priority that can be set in software. If two DMA requests with the same priority occurs at the same time, the one with the lowest channel number will get access to the bus. The Cortex-M3 CPU bus access and the DMA transfers are interleaved in such a way that 60 % of the time on the bus is guaranteed for the Cortex CPU. This means a continuous DMA transfer can maximally use 40 % of the bus bandwidth. This guarantee is important for applications that requires deterministic behavior.

### 4.2.5 Memory Map

The STM32F103 series has a 4 GB linear address space that contains program memory, data memory, registers and I/O ports. The memory map is a good reference when working with this microcontroller and is shown in figure 4.2.

### 4.2.6 Flash

Flash is a non-volatile memory that can be electrically erased and programmed. The STM32 has embedded flash memory that is 16-bit aligned. This means that the flash memory can only be programmed 16-bits at a time and only at every second byte. The flash uses the HSI clock for write and erase operations hence this must be enabled. The flash is divided into pages. A page is a block of memory and is the smallest piece of flash memory that can be erased. On the STM32F103VC used in this thesis, each page corresponds to 2 KB of memory and there are 128 pages in the main memory block. Each page can endure minimum 10 000 erase cycles. (STM32, 2011b, p. 63)

STM32 also support flash write and/or read protection with a special key that has to be written to a specific register to unlock and be able to write and/or read from flash.

### 4.2.7 Standard Peripheral Library (SPL)

SPL is an open source library released by STMicroelectronics for use with STM32 microcontrollers. For STM32F103VC the STM32F10x SPL library is used. SPL is a collection of data structures, macros, functions and examples for the STM32 peripherals. This shortens the development time by abstracting the developer slightly from register level programming for the most part. This library is widely used in this thesis and it is recommened to get familiar with its sematics before looking at the microcontroller code.

### 4.2.8 Register names and bit definitions

All register names are found in their corresponding C structure in the stm32f10x.h header file, as well as bit definitions for the different registers. For example, register names for the USART can be found in the USART structure named USART_TypeDef. Here one can find the USART registers SR, DR, BRR, CR1, CR2, CR3 and GTPR. All USART peripherals on a device has a macro that is a USART_Typedef pointer to the corresponding memory location of the peripheral. Accessing one of these registers in code, e.g., USART1 is done

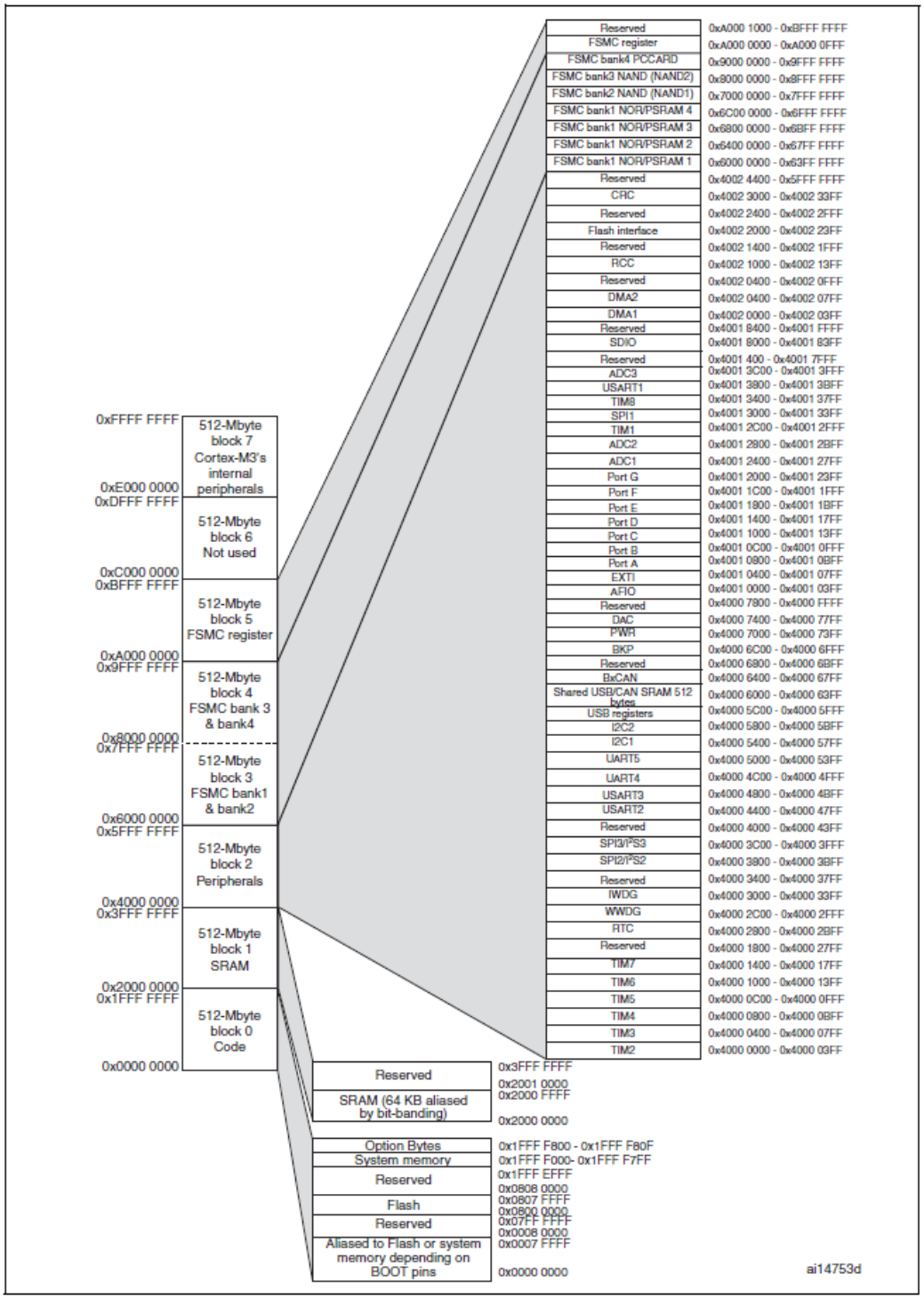



Figure 4.2: STM32 memory map. Taken from (STM32, 2011b, p. 38)

by using the arrow operator, which will dereference the pointer to the structure and access the value of the member. For example, setting the USART1 data register (DR) to 54 hexadecimal would look like this:

$$USART -> DR = 0\text{x}54$$

One can use the bit definitions instead of the cryptic and harder to maintain way of setting bits by left shifting. Below is an example setting the USART Enable bit in USART Control Register 1. One can easily determine which method is more readable.

Left shifting:

$$USART1 -> CR1 \mid= (1 << 13)$$

Bit definition:

$$USART1 -> CR1 \mid= USART\_CR1\_UE$$

One can also use the "set bit" macro provided in the same file:

$$SET\_BIT(USART1 -> CR1,\ USART\_CR1\_UE)$$

## 4.3 Microcontroller

### 4.3.1 Debugging

When debugging on STM32, it is important to properly set how I2C, watchdog, timers and more, should behave. If the watchdog is enabled when debugging it will most likely time out and reset the system when hitting a breakpoint. This is in most cases unwanted behavior and interrupts the debugging process. The I2C bus can timeout when hitting a breakpoint and so on. It is possible to set how some microcontroller peripherals should behave when hitting a breakpoint. If using a watchdog or timer one can set it to stop counting when at a breakpoint, if wanted. This will avoid the debugging process to be the cause of the watchdog time out. More on this can be found in the specific STM32 microcontroller reference manual under debug support.

#### Using the debugger

The debugger makes it possible to monitor the programs flow of execution and watch values of variables and registers at run-time. This helps locating and

remove logical errors from the program. When debugging it is possible to halt the processor at all executable lines of code, giving full control of execution. Breakpoints are used to mark which line of code one wants to investigate closer. It is possible to use one or more breakpoints. If a line with non-executable code is selected, for instance a comment, the next executable line is selected automatically instead. After inserting breakpoints, the debugger can be run. There are usually five commands to control the flow of execution:

1. Step: This command steps to the next executable line of code. If the next line is a function call, one will be taken to the first line in the function.

2. Step Over: This command acts like a step as long as the next line is not a function call. If the next line is a function call, it will run past all the code in that function.

3. Step Out: This command executes the rest of the code in a function then continues debugging the next executable line in the function caller's code.

4. Run to Cursor Line / Continue: This command executes the code until the next breakpoint is reached.

5. Stop: Stops the debugging.

### 4.3.2 Interrupts

An interrupt signals an internal or external event in a microcontroller that requires handling. An interrupt will disrupt the flow of execution. Handling is done in Interrupt Service Routines (ISR) and for USART1 an ISR can look like in figure 4.3:

It is necessary to check the USART 1 interrupt status flags to determine why the interrupt handler have been called, as all USART 1 interrupts will call the same handler. This handler simply checks if a USART receive interrupt have been received and if so, data is pushed to a queue and the USART receive interrupt is cleared.

### 4.3.3 printf()

The printf() function provided by the C standard library, defined in the stdio.h header file is the standard way of printing text when programming in the C language. When printing on, e.g., the Windows operating system, the standard output stream is set by the operating system.

```
void USART1_IRQHandler(void)
{
	//Check if a receive interrupt was triggered.
	if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
	{
		//Get byte and write to queue.
		Q8_FifoWrite(USART1_GetReceiveQueue(), USART_ReceiveData(USART1));

		//Clear USART1 receive data interrupt flag.
		USART_ClearITPendingBit(USART1, USART_IT_RXNE);
	}
}
```

Figure 4.3: ISR code example.

### Retarget printf() to UART

When working on an embedded system without an operating system, like an ARM microcontroller, there is no standard input or output stream. This means one have to implement some low level code usually provided by the operating system. This is called retargeting and often it is retargeted to UART or some other communication bus. In CooCox IDE one can include the newlib syscalls.c file which contains mostly empty function definitions that have to be implemented. The _write() and _read() functions are used by the stdio.h header file for input and output of data. To be able to use printf() one would have to implement _write() and to use scanf() one would have to implement _read(). Only the _write() function has been implemented in this thesis. It was desired to retarget the printf() function to UART and to be able to do so one would have to send one and one byte over UART until all data provided to printf() has been sent. How this was done can be seen in the syscalls.c file in the microcontroller appendix.

### Use printf() with floating point numbers in CooCox

By default, it is not possible to use the printf() function to output floating point numbers in a CooCox project to this date, so a work-around is needed. This can be done in three steps:

1. Enable the use of math.h.
2. In the startup code in startup_stm32f10x_hd.c the line:

   $$(void*)\&pulStack[STACK\_SIZE - 1]$$

   has to be replaced with:

$$(void(*)(void))((unsignedlong)pulStack + sizeof(pulStack)),$$

This is because the stack must be double-word (64-bit) aligned (ARM, 2012, p. 17), hence the cast to unsigned long which will ensure that the stack is accessed properly.

3. Increase the stack size from the standard 1 KB to 16 KB by changing the line:

$$\#define \quad STACK\_SIZE \quad 0x00000100$$

to

$$\#define \quad STACK\_SIZE \quad 0x00001000$$

this must most likely be done to be able to compile, because of the bigger memory footprint needed to include the full printf() implementation.

### 4.3.4 Using standard library math.h in CooCox

To be able to use the functions declared in the C standard library math.h header file with CooCox, one have to link to the library 'm'. This can be done by simply adding the letter 'm' in the project configuration under linked libraries.

## 4.4 Digital Signal Processing (DSP)

### 4.4.1 Sampling theorem

A discrete digital system can only represent a time continuous analog signal by taking samples of it at fixed intervals. To be able to reconstruct and unambiguously determine the frequency content of the signal sampled, the sampling frequency ($f_s$) must be carefully selected. The Nyquist-Shannon sampling theorem states that the maximum frequency measured cannot be larger than half the sampling frequency.

This relation is written as:

$$\frac{f_s}{2} > B \tag{4.1}$$

where $B$ is the bandwidth of the signal.

The left side of equation 4.1 is called the Nyquist frequency.

If this theorem is violated, aliasing will occur. Aliasing means that frequencies above the Nyquist frequency will fold on the frequencies below the Nyquist frequency making it impossible to reconstruct an unambiguous signal. The aliased frequency can be found as:

$$f_a = f - nf_s \tag{4.2}$$

where $f_a$ is the aliased frequency, $f$ the actual signal frequency and $n$ is an integer as large as possible, but also satisfying:

$$nF_s < f \tag{4.3}$$

### 4.4.2 Undersampling

Undersampling (also called sub-sampling) is sampling below the Nyquist frequency. When this is done the absolute frequency information is lost as the signal is aliased (or folded) on the first Nyquist zone from 0 to $\frac{f_s}{2}$.

In 4.4 two signals with the same amplitudes are shown. The blue and red signal has a frequency of 1 Hz and 19 Hz respectively. The sampling rate is 20 Hz, which means the 19 Hz signal is folded on to the first Nyquist zone as a 1 Hz signal. This can be seen as the samples of the signals is marked with red and blue circles. The 20 Hz sampled 19 Hz signal seems to have the same frequency as the 1 Hz signal, but out of phase.

In 4.5 the reconstructed signals can be seen and they both have the same frequency. The 1 Hz signal is correctly reconstructed, while the 19 Hz signal is reconstructed as a inverted 1 Hz signal. Since both signals has the same frequency is it now impossible to distinguish the aliased from the non-aliased signal.

### 4.4.3 Equivalent Time Sampling

For example by using a zero crossing detector (ZCD) to detect when a sinusoidal wave crosses some potential it is possible to time the samples. The ZCD will always trigger at same potential and/or edge and a delay before the sample is taken can be added for example by a hardware timer. This means that if the signal measured on is periodic for some time it is possible to sample consecutive periods of the sine wave and use the samples as if one period with a higher sampling rate was sampled.

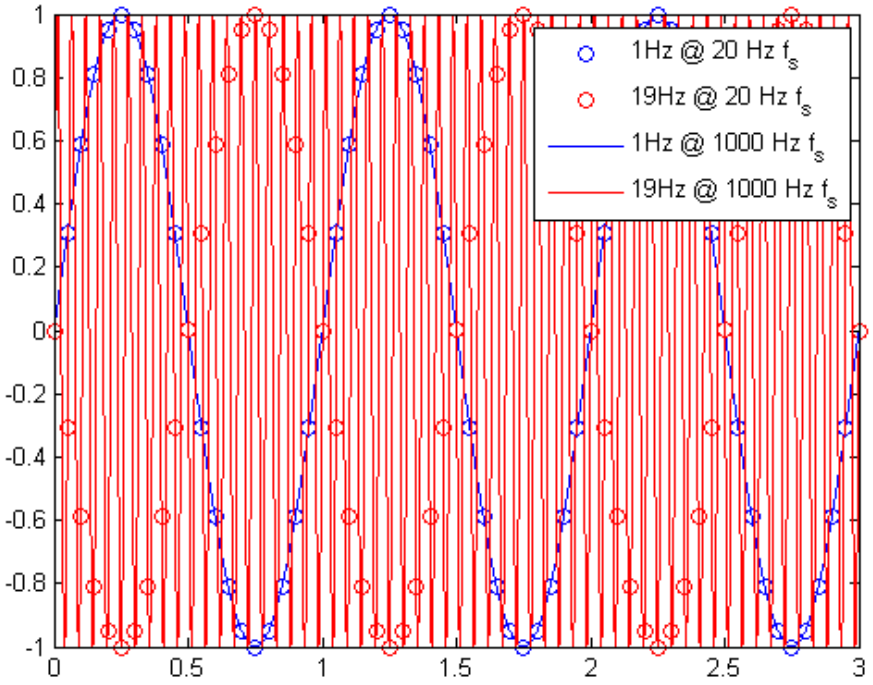



Figure 4.4: A 1 Hz and 19 Hz signal sampled at 20 Hz where the red and blue dots shows the 20 Hz samples respectively.

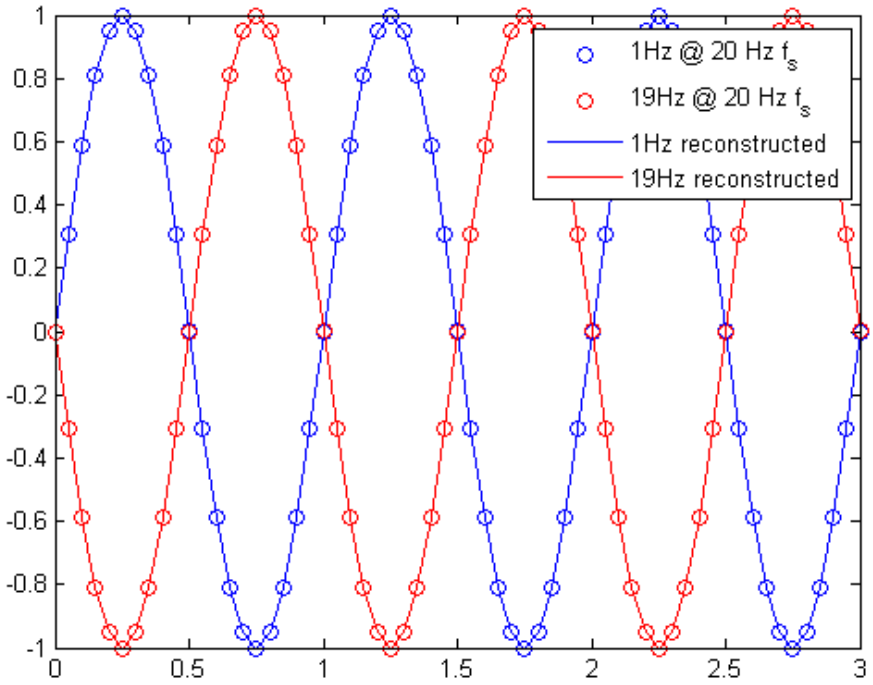



Figure 4.5: The reconstructed 1 Hz and 19 Hz signal

This can be used to sample periodic signals that would require a sampling rate much higher than the chosen ADC. In 4.6 10 periods of a 1 kHz signal is shown in blue. Eleven samples are taken over 11 periods and is shown as red circles. The first sample is taken in the first period when the ZCD is triggered. The next sample is then taken the next time the ZCD triggers, but with a delay of:

$$delay = \frac{1}{11} \cdot \frac{1}{f} \cdot (n-1) \tag{4.4}$$

where $n$ is the period the sample is taken.

To properly see that the collected samples is similar to sampling the 1 kHz signal at 11 kHz all the samples time axis values can be scaled by dividing with $periods + 1$. This is shown as the green circles.

Instead of using a ZCD it is possible to calculate the sampling frequency needed to sample $N$ samples over $M$ periods when a signal has the frequency $f$:

$$\frac{N}{M} = k \tag{4.5}$$

where $k$ is a coefficient that is used to find the sampling frequency:

$$f_s = kf \tag{4.6}$$

If the sampling frequency $f_s$ is used with no ZCD, the signal will be sampled with $N$ samples over $M$ periods and when divided with $periods + 1$ the resulting sine wave is equivalent to as if the signal was sampled with $Mf$ Hertz.

As long as the signal is periodic, ETS can be used to achieve what looks like a very high sampling rate from a slow ADC.

### 4.4.4 Finite Impulse Response Filter

Finite Impulse response (FIR) filter is one of two main types of digital filters, where the other is Infinite Impulse Response (IIR) filter.

A FIR filters output is a weighted average of the $n$ most recent input samples. A FIR filter has $N$ coefficients (often referred to as 'taps' in DSP) and an input signal consisting of a single 1 trailed by zeroes going through the filter will result in a delayed output of the filters coefficients in their correct order. This is called the impulse response. Opposed to IIR filters the FIR filters has a finite impulse response of length equal to number of taps.

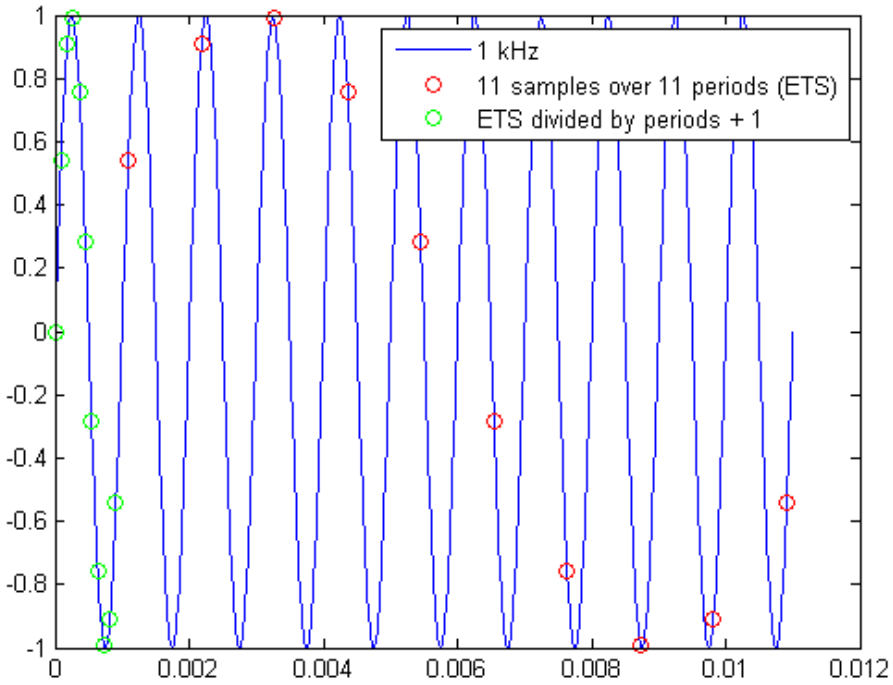



Figure 4.6: An example of an ETS signal.

A FIR filter is always BIBO (Bounded-Input Bounded-Output) stable. This means it will for every bounded input, have a bounded output. This is important in many applications and especially control theory.

Linear phase can be achieved by ensuring that the taps are symmetric around the center tap. If number of taps is even the two most inner taps are equal and if number of taps is odd the taps are symmetrical around the center tap. Linear phase means the phase is a linear function of frequency. All frequency components are shifted in time by the same amount. This time shift or delay is:

$$\text{delay} = \frac{N-1}{2F_s} \tag{4.7}$$

where $N$ is the number of taps and $F_s$ the sampling frequency.

The gain of an $N$-length FIR filter at the angular frequency $\omega$ is:

$$G = \sum_{k=0}^{N-1} h[k]e^{-j\omega k} \tag{4.8}$$

where $h[k]$ are the taps.

It is then possible to normalize the filter so that the gain at the chosen frequency is 1. This is done by dividing all the filter taps by the gain $G$. At DC the gain is simply the sum of the coefficients.

Many types of FIR filters can be constructed. Some of these are low-pass, high-pass, band-pass and band-stop filters. Also other filters performing digital operations such as the Hilbert transform which corresponds to phase shifting a signal by -90 degrees.

The more coefficients used the steeper the transition band will be, but more delay is added and computing power needs are increased. This trade-off must be evaluated for the specific application.

The implementation of a FIR filter can be summarized in three steps:

1. Insert a sample at the input.
2. Multiply each sample with the filter coefficient at its position and accumulate the result, which is the output of the filter.
3. Shift all the samples in the filter by 1 and repeat these three steps.

### 4.4.5 Digital Lock-in Amplifier

Digital lock-in amplifier (also called synchronous detection) is a widely used technique for retrieving a signal buried in noise. The central part of the

system involves a signal that is multiplied (mixed) by a reference signal with the same frequency that is both in-phase and 90 degrees out of phase (the quadrature component). This is called demodulation and the signal is extracted in two components.

The two components are called $I$ and $Q$ and must be low-pass filtered to remove the $2f$ frequency, which is explained later. This process cause the lock-in to focus on the signal exactly at the reference signal frequency and ignore the rest of the frequencies.

The reference signal can be either internal or external in the receiver. In 4.7 an external reference is used.

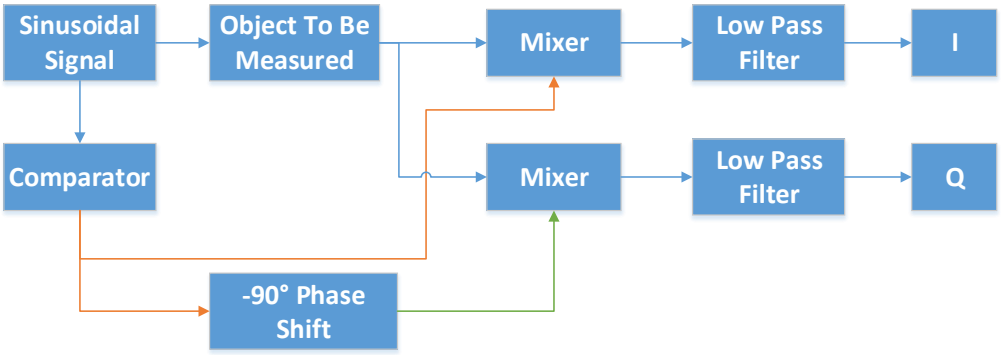



Figure 4.7: Block diagram of a digital lock-in amplifier.

If we have a signal $S(t)$ and a square reference signal $R(t)$ both with no DC and the reference signal has no phase offset and an amplitude of 1:

$$S(t) = A_s \sin(\omega_s t + \theta) \tag{4.9}$$

$$R(t) = \sin(\omega_r t) \tag{4.10}$$

When mixing these two signals we get:

$$M(t) = S(t) \cdot R(t) = A_s \sin(\omega_s t + \theta) \sin(\omega_r t) = \tag{4.11}$$

$$\frac{A_s}{2} \cos[(\omega_s - \omega_r)t + \theta] - \cos[(\omega_s + \omega_r)t + \theta]$$

Since the angular frequencies are equal, we get:

$$\omega = \omega_s = \omega_r \tag{4.12}$$

$$M(t) = \frac{A_s}{2}\cos(\theta) - \cos(2\omega t + \theta) \tag{4.13}$$

After the mixing of the signal and reference signal, the result is a DC component with half the amplitude of the signal amplitude that varies with the cosine of the phase difference and a signal with twice the frequency of the original signals.

If the component at twice the signal frequency is attenuated using a low pass filter, $M(t)$ is a DC signal whose amplitude only varies with the cosine of the phase difference between the two signals.

The in-phase and quadrature-phase signal is then:

$$I(t) = \frac{A_s}{2}\cos(\theta) \tag{4.14}$$

$$Q(t) = \frac{A_s}{2}\cos(\theta - 90°) = \frac{A_s}{2}\sin(\theta) \tag{4.15}$$

Here I(t) is the real part and Q(t) the imaginary part of the signal. The amplitude and phase can be calculated:

$$|A(t)| = \sqrt{I(t)^2 + Q(t)^2} \tag{4.16}$$

$$\phi(t) = \arctan\left(\frac{Q(t)}{I(t)}\right) \tag{4.17}$$

### 4.4.6 4fs Technique

A similar technique that can be used is setting the sampling rate $f_s$ to four time the signal frequency:

$$f_s = 4f \tag{4.18}$$

A signal is sent through a comparator and through the object to be measured. The output of the comparator is used as the reference signal. A zero crossing detector (ZCD) ensures that the signal is sampled at the same point as referenced by the reference signals rising or falling edge. The frequency is known so a time delay corresponding to one fourth of the period can be calculated and a timer is used as a trigger to the sampling ADC. This results in a digital signal $s[n]$ that is the sampled signal $s(t)$ at:

$$t = \frac{n\pi}{2} \qquad \text{where} \qquad n = 0, 1, 2, 3... \tag{4.19}$$

The in-phase component $I$ and quadrature component $Q$ is then:

$$I(t) = s[0] - s[2] \tag{4.20}$$

$$Q(t) = s[3] - s[1] \tag{4.21}$$

This technique is discussed in (Cope, 2003). This technique is computationally a lot more efficient than what was shown in the previous section, which required multiplication of the signal and the reference and filtering. If a FIR filter is used it implies many multiply-and-accumulate (MAC) operations. Instead this technique utilizes a ZCD, a timer and two subtractions.

The calculated phase from I(t) and Q(t) of equation 4.20 and 4.21 is $-90°$ when it should be $0°$. This is an offset and can simply be adjusted for adding a constant of $90°$. Also using the falling or rising edge of the ZCD might offset the signal with $180°$ and can be adjusted for equally.

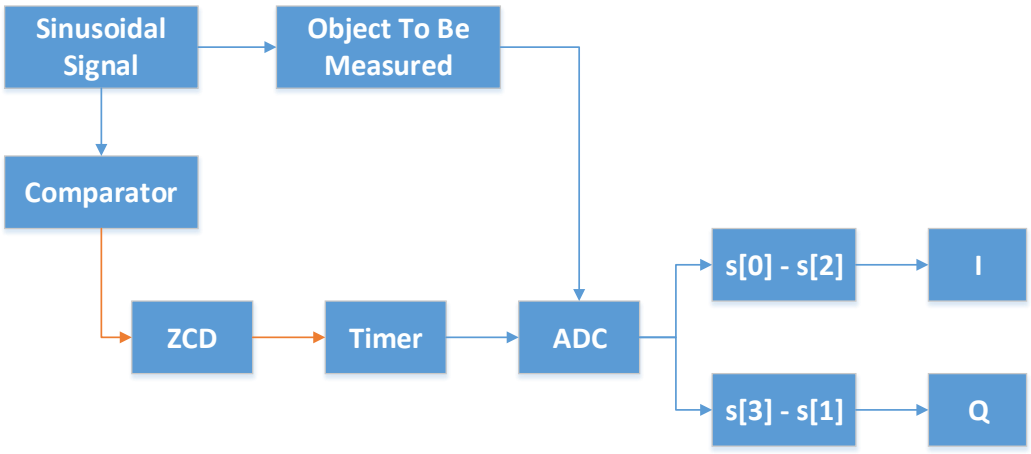



Figure 4.8: Block diagram of a digital lock-in amplifier using this technique.

Maximum frequency is mainly determined by the resolution of the timing reference used.

### 4.4.7 Signal Averaging

If the signal is periodic, the noise is random with a mean of zero and the signal and noise are uncorrelated, it is possible to average the measurements to increase the signal to noise (SNR) ratio. The signal will be added coherently and the noise will not.

The noise will decrease with the square root of the number of measurements as shown in equation 4.22.

$$N = \frac{\sigma}{\sqrt{n}} \tag{4.22}$$

where $\sigma$ the standard deviation of the noise and $n$ the number of measurements.

## 4.5 Fixed-point numbers

Fixed-point numbers is one way to represent a number. The representation is often denoted Qm.n or Qm.f where m is the mantissa and n/f is the fractional part. In addition, the notation Qn or Qf is used. For example Q15 implies that 15 bits are reserved for the fractional part and that one bit is the sign bit. The minimum size of the integer variable needed to store this number is 16 bit. In computing, an integer can not hold a fractional number directly. To be able to do this one would have to reserve some of the integer bits for the fractional part on the expense of the mantissa. The radix point is what splits the mantissa from the fractional part. The radix point is just a concept and cannot be seen in the integer variables data. This means one would have to, at all times, know how many bits are reserved for the mantissa and the fractional part in the integers one wish to use. A scaling factor can be used, which multiplied with the fixed-point number will give the number without the fractional part.

Fixed-point numbers are often very computational efficient compared to floating point numbers. Especially if no floating point unit (FPU) is present, which often is the case with many microcontrollers. On ARM microcontrollers in the Cortex-M series, only the Cortex-M4 has a FPU. By replacing floating point operations by fixed point operations, the computation will often be increased manyfold.

In this thesis, fixed-point numbers are used to speed up the process of digital filter computation. Digital filters are often a recursive and very computational intense process.

## 4.6 Software Tools

### 4.6.1 Software revision control system

Any software project, no matter how small, will benefit from having a revision control system to manage the source code and other files belonging to the project. A revision control system will ensure that current and previous versions of files in the software project is stored in a repository. The user

can add, modify or delete files and commit the changes to the repository. The revision is incremented by one for each successful commit. It will then, at any time, be possible to retrieve an earlier revision of the entire software project. This task would be almost impossible elsewise. Another much-valued functionality is the ability to compare specific files in the software project to older revisions of the same file to see what changes was made. During this thesis, the free open source application TortoiseSVN was used as the software versioning and revision control system. TortoiseSVN provides an easy to use graphical user interface and is available on Windows and Mac OS X.

### 4.6.2 Differencing and merging tool

Even though TortoiseSVN has incorporated its own differencing and merging tool, it also supports adding others. During this thesis WinMerge was used. This tool can be used to compare files, folders and merge files when conflicts occurs during committing to repository or updating working copy.

### 4.6.3 MathWorks MATLAB

MATLAB is a high-level language and development environment used for numerical computation, visualization and programming. MATLAB have been used to test and simulate algorithms and visualize the result. Especially during testing digital and analog signal processing techniques and algorithms. MATLAB is used as it provides a faster way to prototype and visualize some part of the software code than other traditional languages like C/C++ or Java.

### 4.6.4 ARM platform development tools

In this thesis, the tools used to write, compile, link, download and debug code for the ARM microcontroller is the free Integrated Development Environment (IDE) CooCox CoIDE and free GNU ARM toolchain.

#### GNU ARM Toolchain

The GNU ARM toolchain includes among others the GNU Debugger (GDB), GCC compiler and C standard library. It is essential to use an ARM toolchain to be able to work with an ARM microcontroller. Other ARM toolchains are available from for example CodeSourcery, Keil and IAR.

#### CooCox CoIDE

CooCox (Cooperate on Cortex) CoIDE is a relatively new and free software development environment for ARM Cortex microcontrollers. All code can be written, compiled, linked, downloaded to microcontroller and debugged through this IDE.

### 4.6.5 Qt

The Qt project is an open source, C++ framework, IDE and more used to develop GUI and applications.

#### Qt framework

The Qt framework is a powerful open source graphical user interface and application framework and have a lot of functionality. Qt is a cross-platform framework written in C++ and some of the supported platforms are Windows, Linux, Mac OS X, Android, iOS, Blackberry, Windows Phone, WinRT and a lot more. Qt also has QML which is a JavaScript superset declarative language for designing GUI, mostly used for mobile development. In this thesis, Qt have been used for all GUI development on both desktop and Android. QML was not used due to QCustomPlot, which is the library used for plotting, not supporting QML. It is possible to choose different compilers for compiling code under Qt. MinGW GCC 4.8 was used for both desktop and Android.

#### Qt Creator IDE

Qt Creator is a cross-platform IDE and is a part of the Qt project. It also has the Qt Designer embedded which is a GUI layout and form builder. Also debugging, examples and help documentation is integrated in the IDE along with a lot of other functionality.

#### Qt and Android

From the release of Qt 5.2, the Android platform was officially supported. This makes it possible to also deploy applications to Android and combine both the Qt Framework and Android's API written in the Java programming language. This makes it possible to use not yet Qt-implemented Android functionality by calling Android API/Java code. Before being able to compile and deploy for Android, four third-party software's are needed. These are:

1. Android Software Development Kit(SDK)

2. Android Native Development Kit (NDK)
3. Java Development Kit (JDK)
4. Apache Another Neat Tool (ANT)

### 4.6.6 Python

Python is an easy to use interpreted programming language with a large set of libraries.

### 4.6.7 Software Version Overview

Table 4.1 shows the softwares used in this thesis and its versions. Only the newest version is shown if an upgrade have been done during the project.

| **Software** | **Version** |
|---|---|
| Qt Framework | 5.3.0 |
| Qt Creator IDE | 3.1.0 |
| Tortoise SVN | 1.8.4 |
| Android SDK | 20131030 |
| Android NDK | r9b |
| Apache ANT | 1.9.2 |
| JDK | 1.7.0.45 |
| CooCox IDE | 1.7.6 |
| ARM Toolchain | 4.8.0 |
| MathWorks MATLAB | 2014a |
| WinMerge | 2.14.0.0 |
| Standard Peripheral Library | 3.5.0 |
| Python | 3.3.0 |
| MatPlotLib | 1.2.0 |
| CMSIS DSP Library | 1.1.0 |
| CMSIS | 3.01 |

Table 4.1: List of software and versions used.

## 4.7 Android / Java

The Android operating system is based on the Linux kernel and is designed for touchscreen phones and tablets. Android applications are primarily written in the Java programming language, but in this thesis the Qt Framework

and the C++ programming language has been mostly used. The exception is the Bluetooth functionality which was not supported by the Qt Framework at the time of development and is therefore written in Java.

### 4.7.1 Android Manifest

The Android Application Manifest must be included in all Android applications to be able to run on an Android platform. It contains essential information about the application to be used by the Android system such as the name of the Java package, application name and which icon to be used. Other information it contains are the permissions needed to be run, minimum and targeted SDK level, libraries that needs to be linked and which mechanisms are used to access the Android operating system and the hardware.

### 4.7.2 Java Native Interface

The Java and Android libraries are both written in the Java programming language. The Java Native Interface (JNI) makes it possible for Java code to be called from C/C++ and the other way around. JNI was used to be able to implement the Bluetooth functionality. Methods from the Android's 'android.bluetooth' package was called from the C++ side of the developed application.

### 4.7.3 Singleton design pattern

The singleton pattern limits the instantiation of a class to one object. This implicitly also means that static methods can get access to the 'this' pointer and the members of the class. This design pattern can be useful in cases where it only makes sense to have one instance. An example of this is if a device only has one Bluetooth hardware available. The singleton pattern made it easier integrating the Bluetooth Java code with C++ through JNI. It does not introduce any limitation for the purpose of the developed application. Both the C++ class calling the Bluetooth Java code and the Bluetooth Java code class was designed as singletons to make sure the objects are synchronized at all times even if more instances of either object was tried instantiated or static functions was called to modify the objects.

## 4.8 Python Serial Port Application

In the earlier stages of the project a application was made in the Python language that would later be replaced by the GUI. The Python application is a simple serial port communication program that can receive and send data and show it in the terminal window. It also uses the MatPlotLib, which is a Python plotting library, to show simple plots of data received over the serial port. This was especially used to visualize sampled data and DSP operations applied to this data. The Python application can be found in the appendix.

# Chapter 5

# System Design: Hardware

## 5.1 Overview

Figure 5.1 is an overview of the hardware platform and their interconnections. The microcontroller communicates with a PC or Android system over the Bluetooth module, controls the Direct Digital Synthesizer (DDS) and receives measurement signals from the analog front-end (AFE).

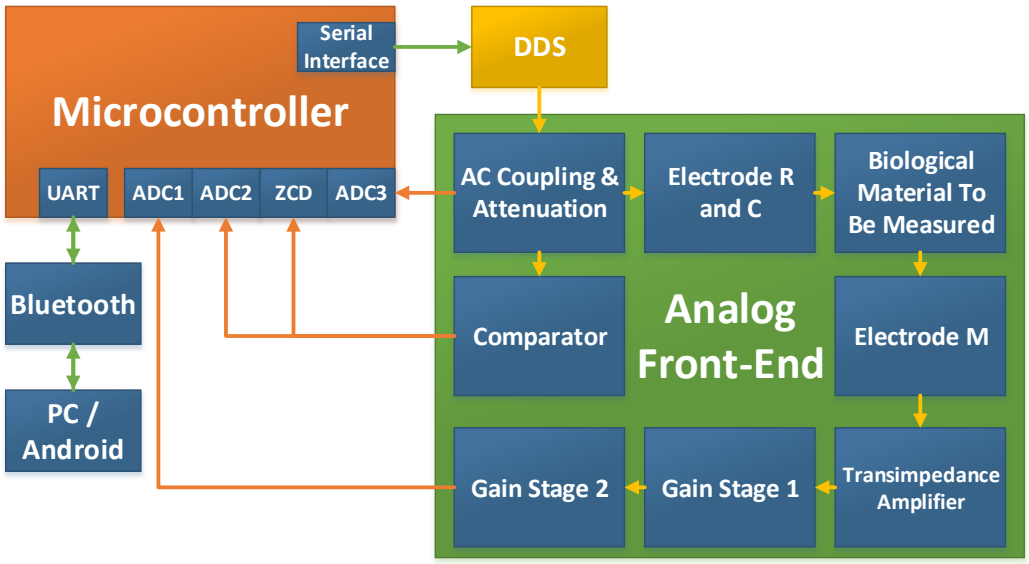



Figure 5.1: Hardware design overview.

The table 5.1 shows the names of the interconnections.

| Microcontroller | DDS | Analog Front-End | Bluetooth |
|---|---|---|---|
| 3.3V | | | Vcc |
| PA9 | | | Rx |
| PA10 | | | Tx |
| 3.3V | Vcc | | |
| PE12 | D7 | | |
| PE14 | W_CLK | | |
| PB10 | FQ_UP | | |
| PB11 | Reset | | |
| 3.3V | | Vcc | |
| | Iout (After filter) | AC coupling & Attenuation in | |
| ADC1 | | Gain Stage 2 out | |
| ADC2 | | Comparator out | |
| ADC3 | | AC coupling & Attenuation out | |
| ZCD | | Comparator out | |

Table 5.1: Hardware interconnection overview.

## 5.2 DDS

The AD9850 integrated circuit (IC) from Analog Devices was chosen as the DDS to use in this prototype.

AD9850 satisfies the requirements as seen in table 5.2 and it was available as an evaluation board with 2.54 mm pitch making it easy to use in prototyping. The evaluation board schematic is shown in the appendix and a picture of the module is shown in figure 5.2.

| Specification | Requirement | AD9850 |
|---|---|---|
| Power | 3.3V Sisngle-Supply | 3.3V or 5V |
| Output frequency | 1 MHz or higher | Up to 55 MHz at 3.3V |
| Digital-to-Analog (DAC) resolution | 10-bit or higher | 10-bit |
| Communication interface | Serial | Serial or Parallel |
| Frequency resolution | 100 Hz or less | 29.1 mHz at 125 MHz clock |

Table 5.2: DDS requirements.

An overview of the input and outputs used can be seen in figure 5.3. $V_{cc}$ and $GND$ are the power and ground line respectively, $I_{out}$ the AD9850's internal DAC output, $RESET$ the reset line and $D7$, $W\_CLK$ and $FQ\_UP$

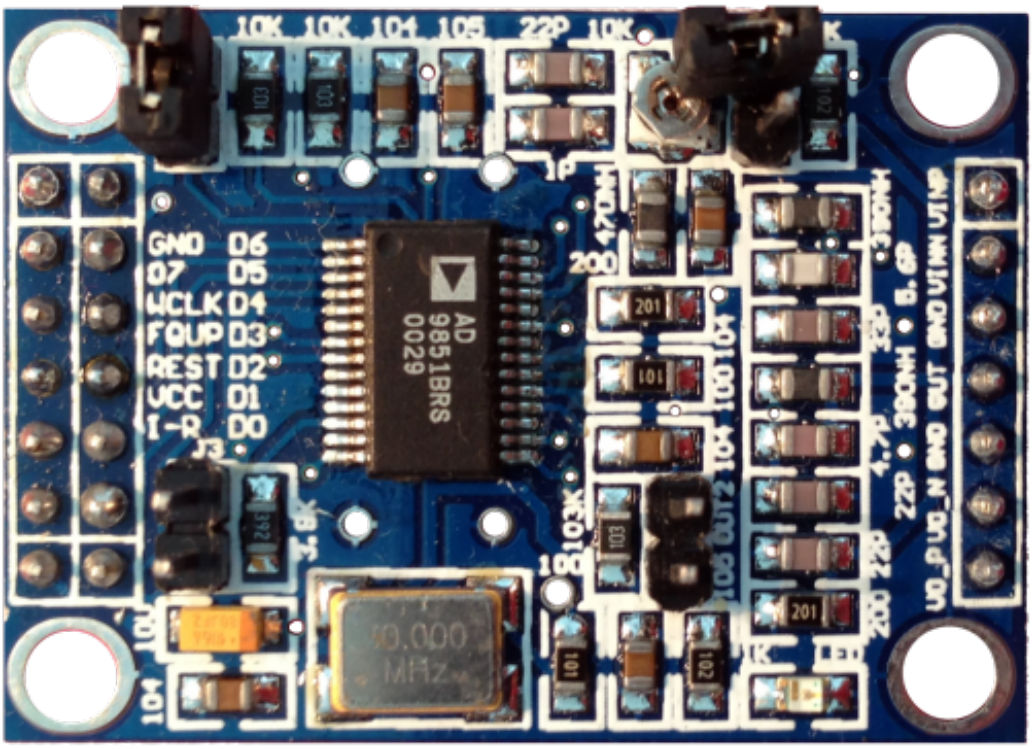



Figure 5.2: An image of the DDS module.

part of the digital serial communication input lines.

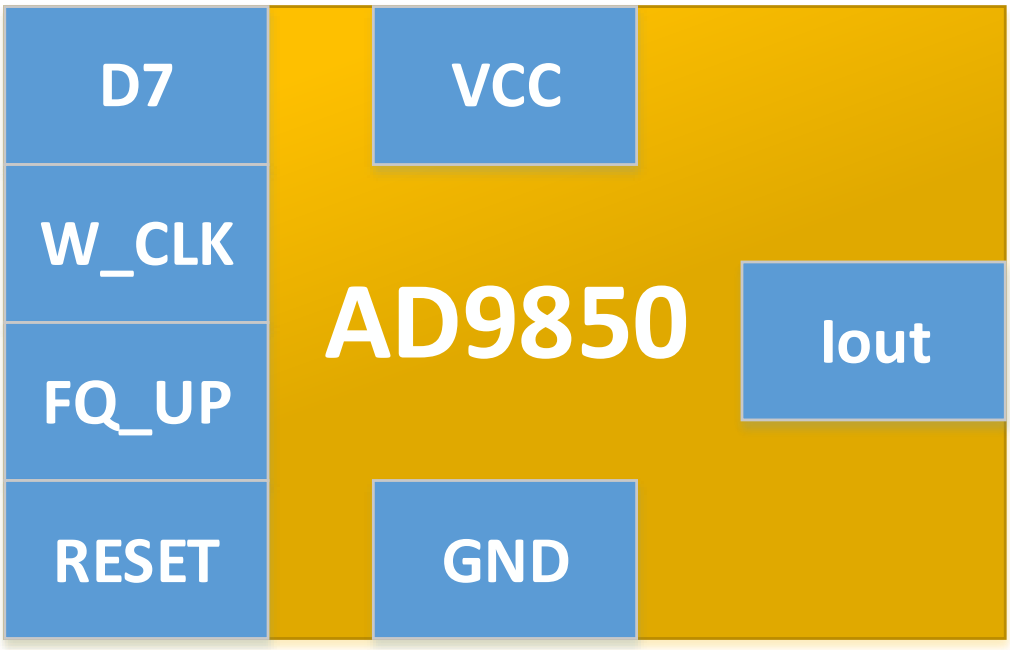



Figure 5.3: DDS input/output overview.

The AD9850 output signal ($I_{out}$) peak-to-peak current is 10 mA and the cascaded filter has an impedance of about 100 Ω. The output signal is then approximately:

$$V_{out} = 10 \text{ mA} \cdot 100 \ \Omega = 1 \text{ V} \tag{5.1}$$

The output signals peak-to-peak values varies with frequency. More specifically, it varies with the envelope of the normalized sinc function:

$$\text{sinc } (\frac{f_{out}}{f_{clk}}) = \frac{\sin{(\pi \frac{f_{out}}{f_{clk}})}}{\pi \frac{f_{out}}{f_{clk}}} \tag{5.2}$$

This is shown in the data sheet and a modified figure is shown in figure 5.4.

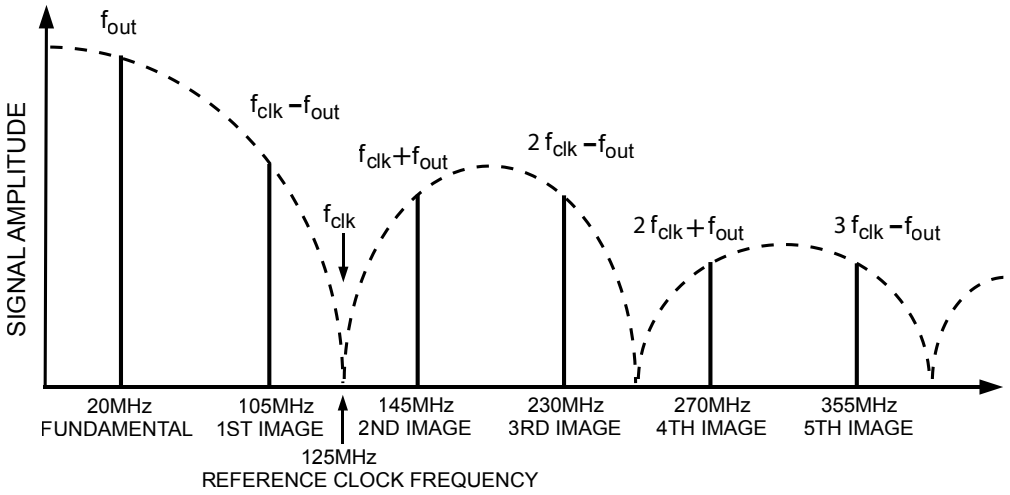



Figure 5.4: DDS signal amplitude as function of frequency.

The AD9850 has a 40-bit register that is set by the microcontroller. The content of the register is shown in 5.5.

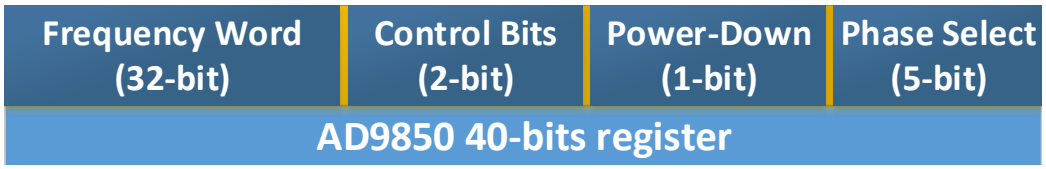



Figure 5.5: The programmable register of the AD9850.

The resolution of the DDS is:

$$\text{resolution} = \frac{f_{clk}}{2^N} = \frac{125 \cdot 10^6}{2^{32}} = 0.0291 \text{ Hz} \tag{5.3}$$

where $N$ is the number of bits of the frequency word.

The value to set to the frequency word in the register for a chosen output frequency is then:

$$\text{register value} = 2^{32} - \frac{f_{out}}{\text{resolution}} \tag{5.4}$$

The output frequency is at its lowest when the register is $2^{32}$ and at its highest when its zero.

The two controls bits are not to be set by the programmer and are both set zero.

The power-down bit is set high to power down the DDS and low to power on.

The phase select bits can be used to shift the phase of the output signal, but these have not been utilized in this project.

When writing to the register over the serial interface, the $D7$ input is read at each rising edge of $W\_CLK$. When all the 40 bits are shifted into the register a pulse on the $FQ\_UD$ input is needed to update the AD9850.

## 5.3 Microcontroller Printed Circuit Board

The STM32F103VC microcontroller printed circuit board (PCB) chosen is "HY-Mini STM32V" manufactured by HAOYU Electronics. It is a break-out board where 3.3V, 5V and all of the general purpose input and outputs (GPIO) are made available by mounting two 2x24 2.54 mm pitch male connection headers on each side of the PCB. An 8 MHz crystal oscillator, that will provide 72 MHz by using the internal Phase-Locked Loop (PLL), runs the system clock. It also has a USB-to-UART translator, SD card connector, a 32.768 kHz crystal oscillator and a JTAG programming and debugging interface.

In addition, a reset and boot selection button as well as two input buttons, a system power and USB power LED, two LEDs hardwired to GPIO pins.

A fixed 3.3V low dropout (LDO) regulator regulates the power from the USB port before reaching the microcontroller. The LDO used is Linear Technologies LT1117, which is guaranteed to source a minimum of 800 mA. A button cell battery connector is provided to power the Real Time Clock (RTC) and the microcontroller's backup registers when the main power supply is off.

The full schematic can be found in the appendix and the PCB is displayed in figure 5.6.

### 5.3.1 ST-LINK/V2 Programmer/Debugger

The programmer/debugger used is the ST-LINK/V2 from STMicroelectronics. It has a JTAG interface and uses USB to connect to the computer. The ST-LINK/V2 is shown in figure 5.7.

## 5.4 Analog front-end

To appropriately prepare the output signal from the DDS to excite the biological material and to measure the response and condition the analog signal

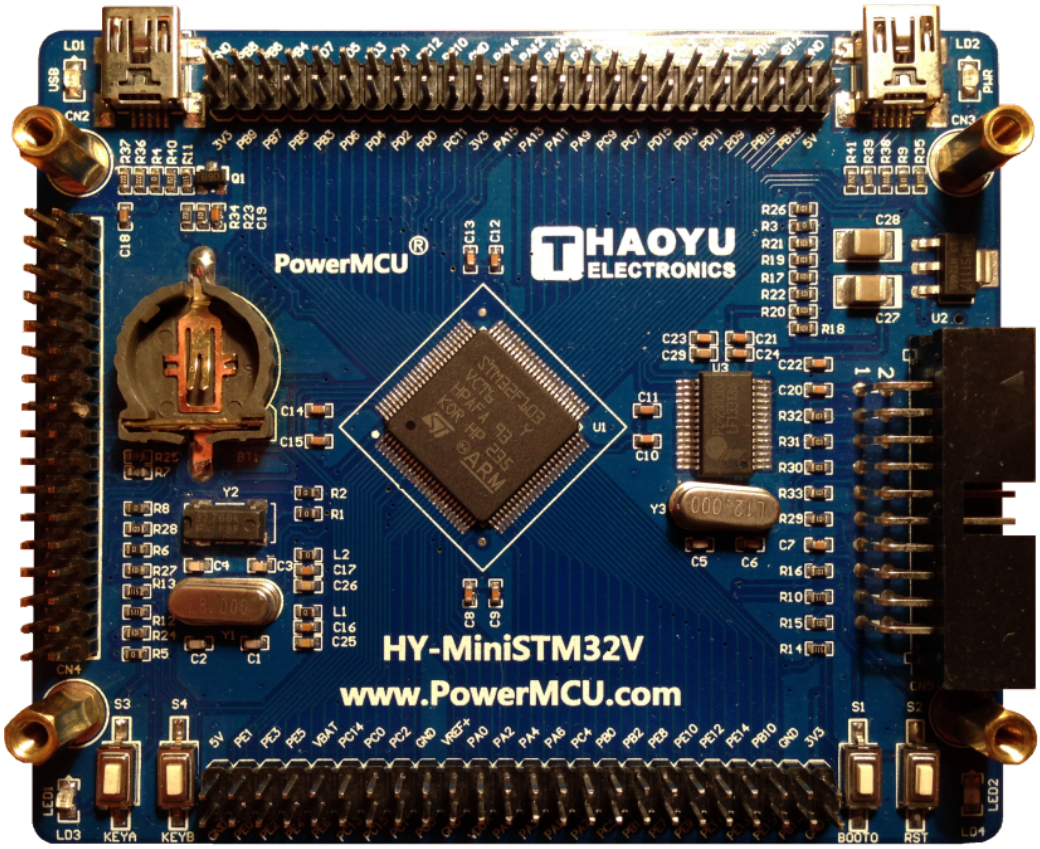



Figure 5.6: Image of the microcontroller PCB.


Figure 5.7: Image of the ST-LINK/V2 programmer/debugger.

for digitizing, an analog front-end is needed.

All operational amplifiers (op-amp) shown in this section are supplied with the microcontrollers 3.3 volts output on their positive power rail and the negative power rail is grounded.

The entire hardware schematic can be found in the appendix.

### 5.4.1 Virtual Ground

This systems virtual ground corresponds to the center voltage between two equal power supplies of opposite polarity in a dual power supply system. This is a common method in single power supply systems and this reference voltage is used to bias the rest of the analog front-end electronics to properly work with a single power supply.

The circuit used is shown in figure 5.8. It uses the OPA350 op-amp from Texas Instruments and has a voltage divider with two equal fixed resistors of 1 kΩ connected to its non-inverting input. Also a 47 nF capacitor is shunted to ground from the non-inverting input. This capacitor is connected to compensate for quick fluctuations in the input voltage.

The output is connected to the inverting input. OPA350 is unity gain stable so no resistor is needed in the feedback loop for stability reasons.

The voltage divider is connected to ground and the microcontrollers 3.3

V power supply.

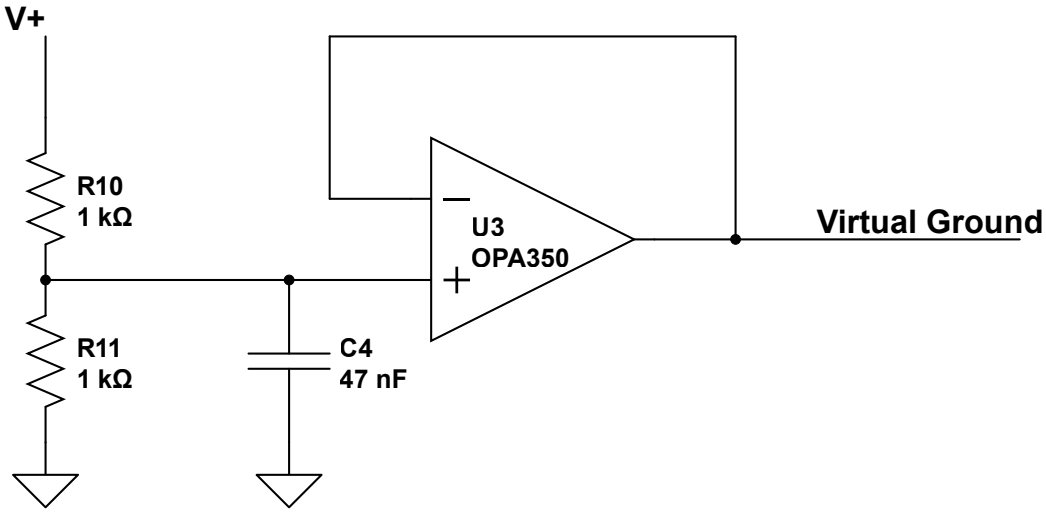



Figure 5.8: Virtual ground circuit used in the hardware prototypes.

The expected output voltage is:

$$V_{out} = \frac{R_{11}}{R_{10} + R_{11}} \cdot V_{+} = \frac{1000\ \Omega}{1000\ \Omega + 1000\ \Omega} \cdot 3.3\ \text{V} = 1.65\ \text{V} \tag{5.5}$$

### 5.4.2 AC coupling and attenuation

The AC coupling and attenuation stage shown in figure 5.9 has three main tasks:

1. Remove the DC offset from the DDS output signal.
2. Attenuate the input signal.
3. Bias the signal to virtual ground.

The input voltage comes from the DDS with a peak-to-peak amplitude of about 1 V as calculated in equation 5.1.

The DDS output signal is first meet by a high pass filter that ensures DC voltage is blocked. The high pass filters cut-off frequency $f_c$ is:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 10000 \cdot 10^{-6}} = 15.92\ \text{Hz} \tag{5.6}$$

The output signal is attenuated by an inverting amplifier where the feedback resistor value is less than the input resistor value. A potentiometer is connected as a variable resistor in the feedback loop.

The virtual ground reference voltage on the non-inverting input ensures that the attenuated AC signal is centered around virtual ground reference voltage on the output.

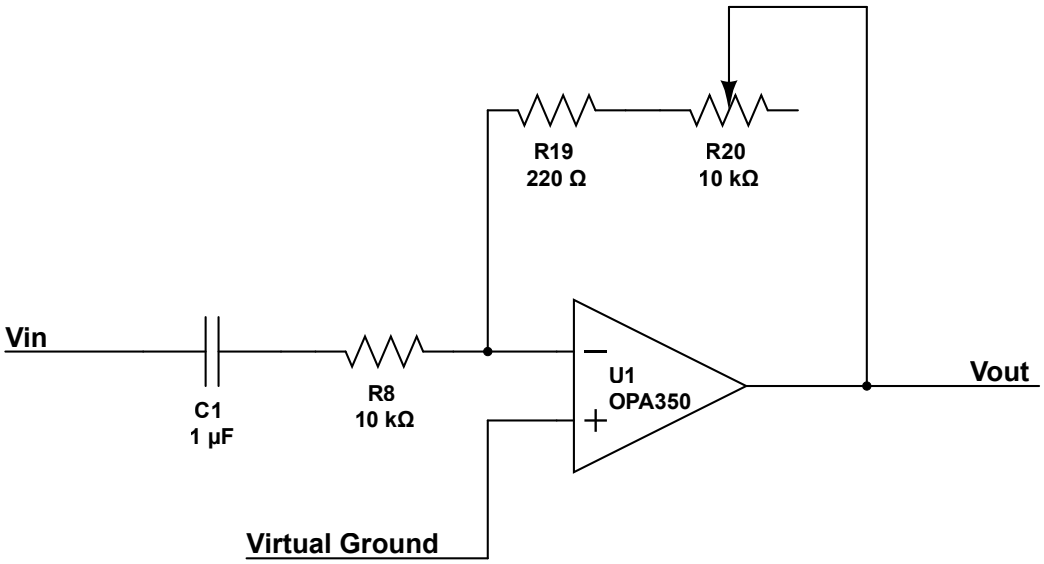



Figure 5.9: The AC coupling and attenuation stage used in the hardware prototypes.

The output for a given input is:

$$V_{out} = \frac{R_{19} + R_{20}}{R_8} \cdot V_{in(AC)} \tag{5.7}$$

Equation 5.7 is only valid for frequencies much higher than the high pass filters cut-off frequency from equation 5.6.

### 5.4.3 Comparator

The chosen comparator was MAX941CPA from Maxim Integrated. It is a 3 V compliant single supply comparator with 80 ns propagation delay and internal hysteresis. It is manufactured in an 8-pin plastic dual inline package (DIP-8) package among others. It also have a low operational current draw of maximum 600 $\mu A$ at 3 V and a shutdown current draw of a few tens of microamperes.

The comparator has the output from the AC coupling and attenuation stage on its non-inverting input and the virtual ground reference voltage on its inverting input.

This means the comparators output is high when the input AC signal on its non-inverting input is above the virtual ground reference voltage and visa versa.

The comparator stage can be seen in figure 5.10.

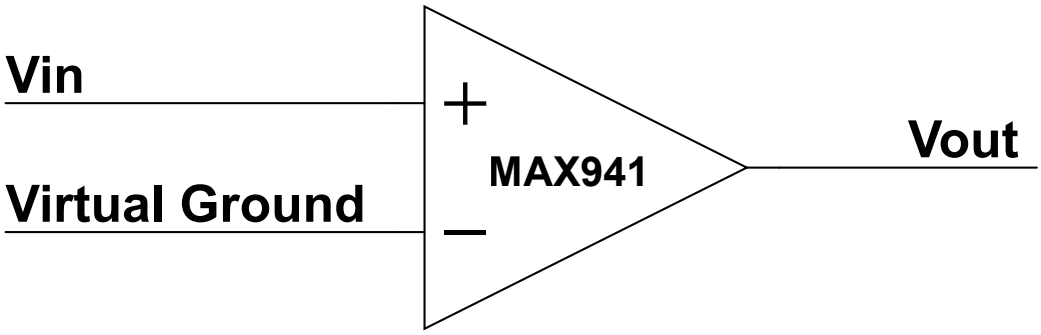



Figure 5.10: The comparator stage used in the hardware prototypes.

### 5.4.4 Three electrode excitation stage

This stage is a buffer using a voltage follower and interface for the reference and counter electrode. The circuit is depicted in figure 5.11.

The op-amp used is the OPA350. The non-inverting input is connected to the output of the AC coupling and attenuation stage. The feedback loops tries to provide the same voltage on the inverting input and within the limits of the op-amp this guarantees that the potential between the R and C electrode is the same as the voltage on the non-inverting input.

### 5.4.5 Transimpedance Amplifier

The OPA350 op-amp was used as the transimpedance amplifier (TIA).

Electrode M is connected to the TIA's inverting input and the non-inverting input is connected to the virtual ground reference voltage. The used circuit is seen in figure 5.12.

The current from the M electrode flows through the feedback resistor, which has chosen to be a 680 Ω fixed resistor in series with a 10 kΩ potentiometer connected as a variable resistor.

OPA350 has a gain bandwidth product (GBP) of 38 MHz and an input capacitance of 6.5 pF in common-mode.

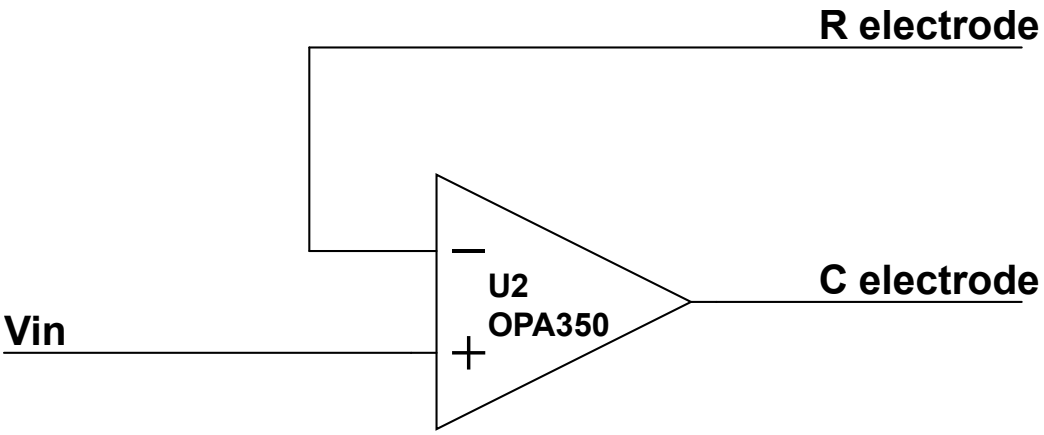



Figure 5.11: The electrode interface stage used in the hardware prototypes.

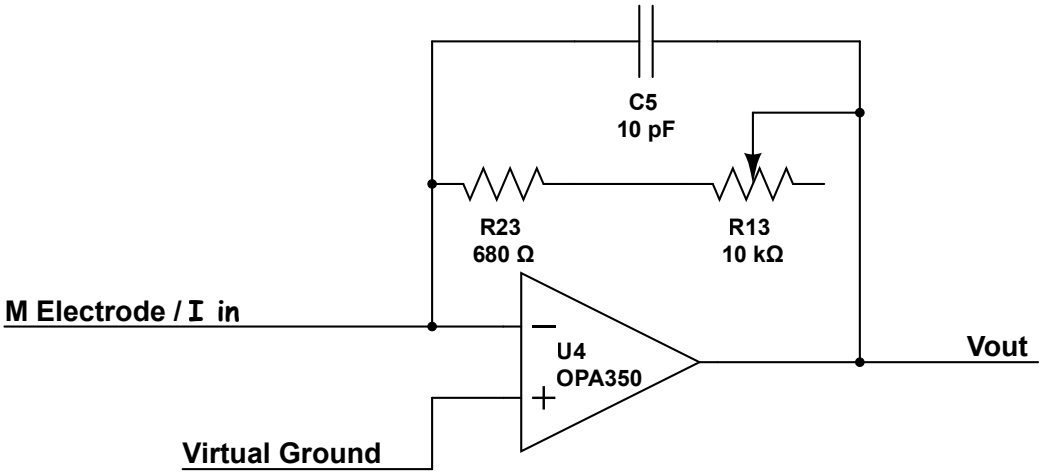



Figure 5.12: The transimpedance amplifier stage used in the hardware prototypes.

Using equation 3.7 the feedback compensation capacitors capacitance can be calculated. The upper and lower limit for the chosen variable resistor is:

$$C_{f\,lower} = \sqrt{\frac{C_s + Ci}{2\pi R_f f_{GBP}}} = \sqrt{\frac{0 + 6.5\text{ pF}}{2\pi \cdot 680\ \Omega \cdot 38\text{ MHz}}} = 6.3pF \tag{5.8}$$

$$C_{f\,upper} = \sqrt{\frac{C_s + Ci}{2\pi R_f f_{GBP}}} = \sqrt{\frac{0 + 6.5\text{ pF}}{2\pi \cdot 10680\ \Omega \cdot 38\text{ MHz}}} = 1.6pF \tag{5.9}$$

the shunt capacitance $C_s$ is set to zero as it depends on what is measured and the values in equation 5.8 and 5.9 is most accurate for values less than the op-amps input capacitance.

A shunt resistor between 1.6 pF and 6.3 pF is recommended. There will be a couple of picofarads stray capacitance and slight overcompensating is shown in equation 5.10 not to be critical in this application so a value of 10 pF has been selected for the feedback capacitor value.

The transimpedance cut-off frequency can then be calculated using equation 3.8. For the maximum value of the feedback resistor the cut-off frequency is:

$$f_{-3dB} = \sqrt{\frac{f_{GBP}}{2\pi R_f C_f}} = \sqrt{\frac{38\text{ MHz}}{2\pi \cdot 10680\ \Omega \cdot 10pF}} = 7.53\text{MHz} \tag{5.10}$$

This is well within the ± 40% variations as explained in chapter 3.3.2.

The OPA350 was also chosen for its low noise parameters. The input voltage noise density is 7 $\frac{\text{nV}}{\sqrt{\text{Hz}}}$ and current noise density is 4 $\frac{\text{fA}}{\sqrt{\text{Hz}}}$.

The total noise contribution from the transimpedance amplifier can be calculated using the equations from chapter 3.12.

The equivalent noise bandwidth can be calculated from equation 3.21 as:

$$\text{ENBW} = f_{-3dB} \cdot \frac{\pi}{2} = 7.53\text{ MHz} \cdot \frac{\pi}{2} = 11.83\text{ MHz} \tag{5.11}$$

Equation 3.22 gives the feedback resistors rms voltage noise:

$$\begin{aligned} N_{Rf} &= \sqrt{4kT \cdot \text{ENBW} \cdot R_f} \\ &= \sqrt{4 \cdot 1.38 \cdot \frac{J}{K} \cdot 298\text{ K} \cdot 11.83\text{ MHz} \cdot 10680\ \Omega} = \underline{45.6\ \mu V_{rms}} \end{aligned} \tag{5.12}$$

The op-amps current noise is given by equation 3.23:

$$N_{current} = I_n R_f \cdot \sqrt{\text{ENBW}} = 4\frac{\text{fA}}{\sqrt{\text{Hz}}} \cdot 10680\ \Omega \cdot \sqrt{11.83\ \text{MHz}} = \underline{0.33\ \mu V_{rms}} \tag{5.13}$$

The plateau/peak noise used to calculate the voltage noise is as given in equation 3.24:

$$N_2 = e_n(\frac{C_f + C_{sh} + C_i}{C_f}) = 7\frac{\text{nV}}{\sqrt{\text{Hz}}} \cdot (\frac{10\ pF + 0\ pF + 6.5\ pF}{10\ pF}) = 11.6\frac{\text{nV}}{\sqrt{\text{Hz}}} \tag{5.14}$$

and the second pole used to calculate the voltage noise as given in equation 3.26 is:

$$f_{p2} = f_{GBP}\frac{C_f}{C_f + C_{sh} + C_i} = 38\ \text{MHz} \cdot (\frac{10\ pF}{10\ pF + 0\ pF + 6.5\ pF}) = 23.03\ \text{MHz} \tag{5.15}$$

The voltage noise is then defined by equation 3.25 as:

$$N_{voltage} = N_2 \cdot \sqrt{\frac{\pi}{2} f_{p2}} = 11.6\frac{\text{nV}}{\sqrt{\text{Hz}}} \cdot \sqrt{\frac{\pi}{2} \cdot 23.03\ \text{MHz}} = \underline{69.8\ \mu V_{rms}} \tag{5.16}$$

The total noise when no or low shunt capacitance relative to the feedback capacitance and the op-amps input capacitance is given by equation 3.27 as:

$$\begin{aligned} N_{total} &= \sqrt{N_{Rf}^2 + N_{current}^2 + N_{voltage}^2} \\ &= \sqrt{45.6\mu V_{rms}^2 + 0.33\mu V_{rms}^2 + 69.8\mu V_{rms}^2} = \underline{\underline{83.4\mu V_{rms}}} \end{aligned} \tag{5.17}$$

The total noise is *very* low, but the noise is amplified in the following gain stages with their gain and also their own inherent noise.

### 5.4.6 Gain stages

The hardware prototypes has two gain stages. A gain stage is illustrated in figure 5.13.

The gain stage is realized with an inverting op-amp configuration. The input resistor is 1 kΩ. In the feedback loop a 680 Ω resistor in series with

a 10 kΩ potentiometer connected as a variable resistor has been selected. The non-inverting input is connected to the virtual ground reference voltage ensuring that the output is centered on this voltage.

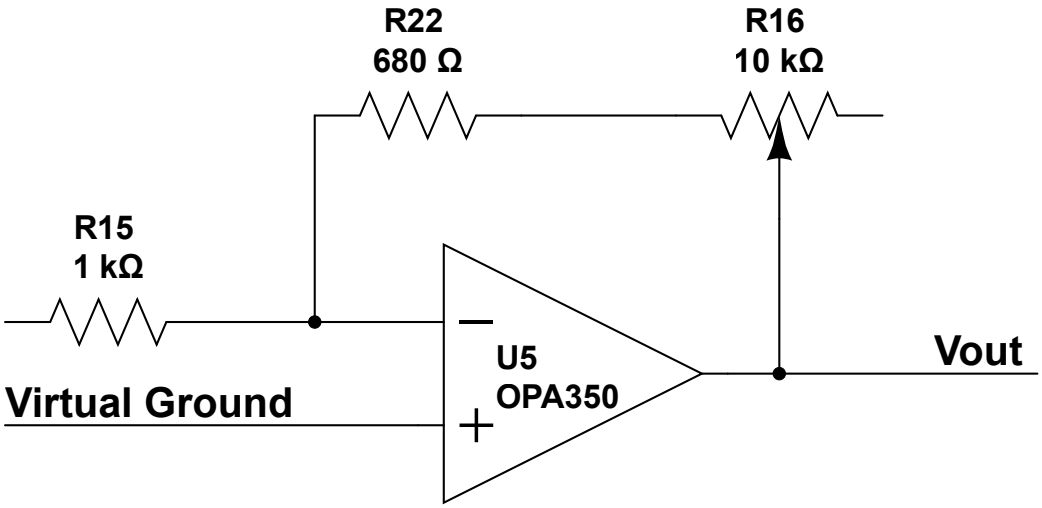



Figure 5.13: The gain stage used in the hardware prototypes.

The output voltage for a given input voltage for the circuit in figure 5.13 is:

$$V_{out} = \frac{R_{16} + R_{22}}{R_{15}} \cdot V_{in} \tag{5.18}$$

### 5.4.7 Decoupling capacitors

A number of decoupling capacitors has been used. Table 5.3 is an overview.

| **Stage** | **Capacitance** |
|---|---|
| Virtual Ground | 47 nF |
| AC Coupling & Attenuation | 47 nF |
| Comparator | 47 nF |
| Transimpedance Amplifier | 47 nF |
| Gain Stage 1 | 47 nF |
| Gain Stage 2 | 47 nF |

Table 5.3: Table of decoupling capacitors used.

### 5.4.8 Simulation

The analog front-end circuit shown in the appendix was simulated using Circuit Lab (www.circuitlab.com). The OPA350's parameters were added to make the simulation more correct. The magnitude and phase plot is shown in figure 5.14. The main area of interest is between 1 kHz and a few hundred kilohertz. Figure 5.14 shows there is close to no change in magnitude from 1 kHz to 100 kHz and the phase is approximately 1° at 1 kHz, −9° at 100 kHz and −51° at 1 MHz. The phase decreases linearly with frequency and this can be calibrated for. Even at 1 MHz, the magnitude has only decreased with 2.2 dBV (factor of 1.3).

### 5.4.9 Prototypes

The first prototype was made on a breadboard using cheap op-amps like the LM741 from Texas Instruments and MCP6002 from Microchip. It did not have variable resistors or decoupling capacitors and was connected with breadboard jumper wires. It was used to test low frequency operations as an early test to be used with the microcontroller. The first prototype is pictured in figure 5.15.

The second prototype was identical to the first except that the op-amps was substituted with the OPA350 op-amp and MAX940CPA comparator. This was used to test on higher frequencies and as a preparatory stage before constructing the third prototype.

The third prototype was made on a prototype board with a 31 x 16 matrix of plated vias. All components was soldered to the board, including the DDS and Bluetooth module. It was now possible to fit the analog front-end board directly on the microcontroller breakout board. The third prototype is shown in figure 5.16.

## 5.5 Bluetooth

For the wireless transmission a Bluetooth 2.0 compatible module named "JY_MCU BT_BOARD V1.4" has been selected. It supports 3.3-5 V logic and a 3.6-6 V power supply. This modules supports the RFCOMM protocol which emulates a serial link over Bluetooth. This means it can be directly interfaced with UART on the microcontroller and the device connected to this Bluetooth module will receive data as if it was a RS-232 interface. The module can be configured using a set of AT commands. The baud-rate should be 9600 kbps by default or can be set using the AT command "AT+BAUD4"

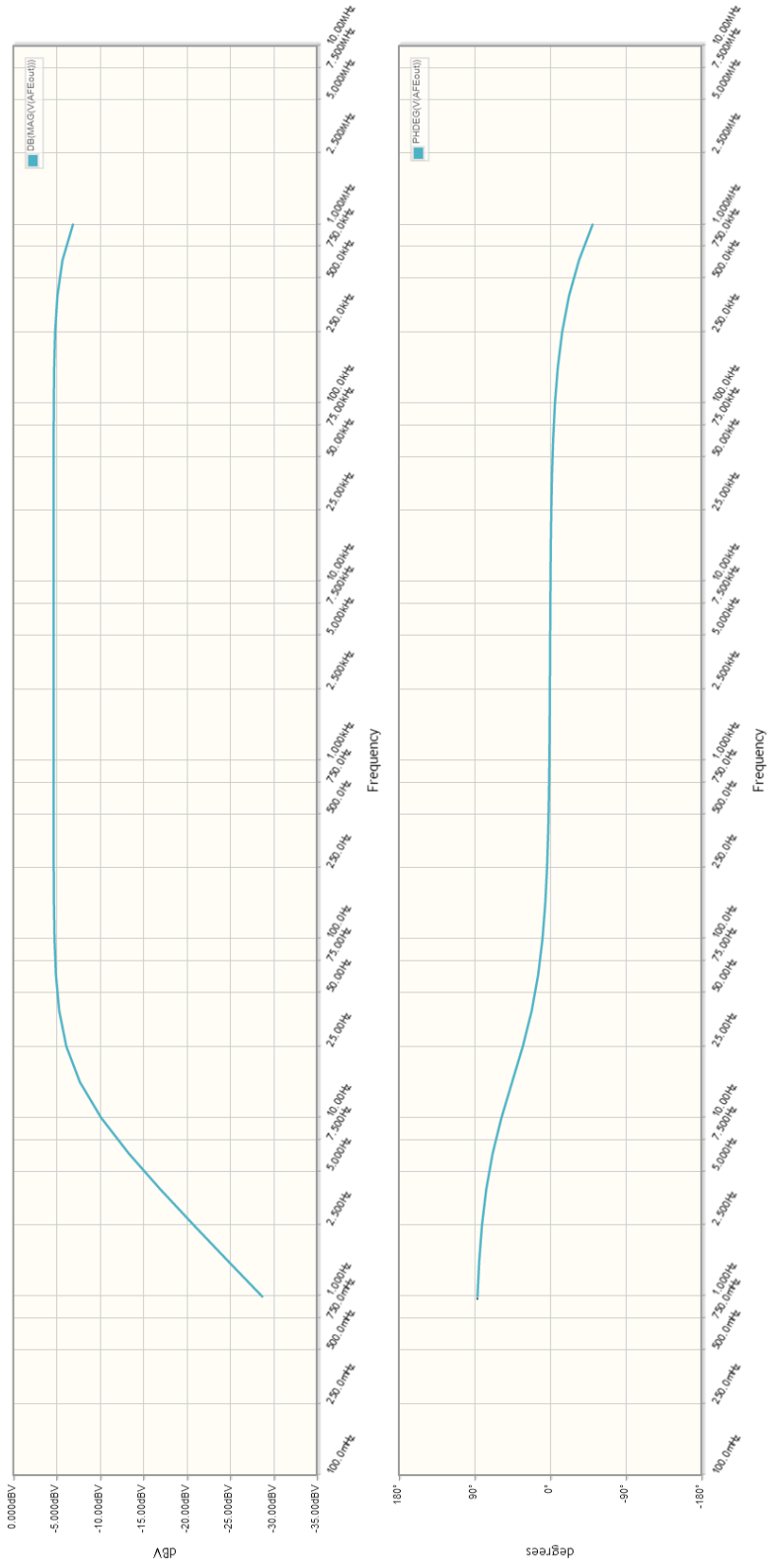



Figure 5.14: Magnitude and phase plot of the simulated circuit.

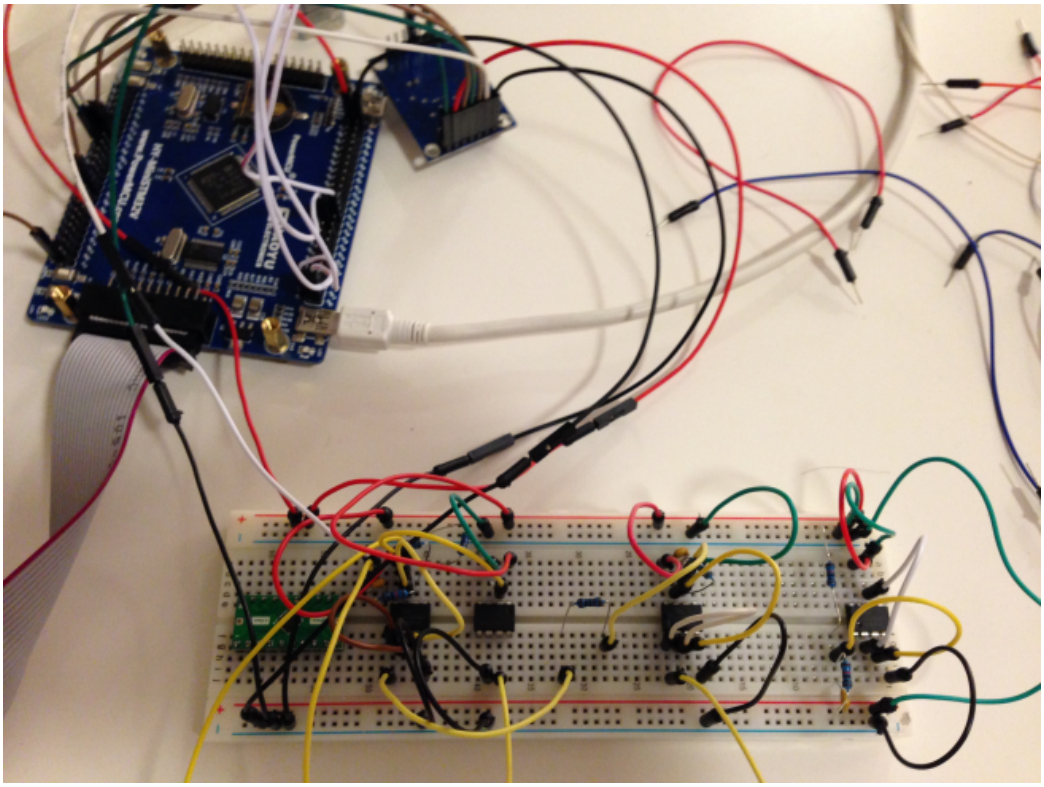


Figure 5.15: A picture of the first/second prototype.

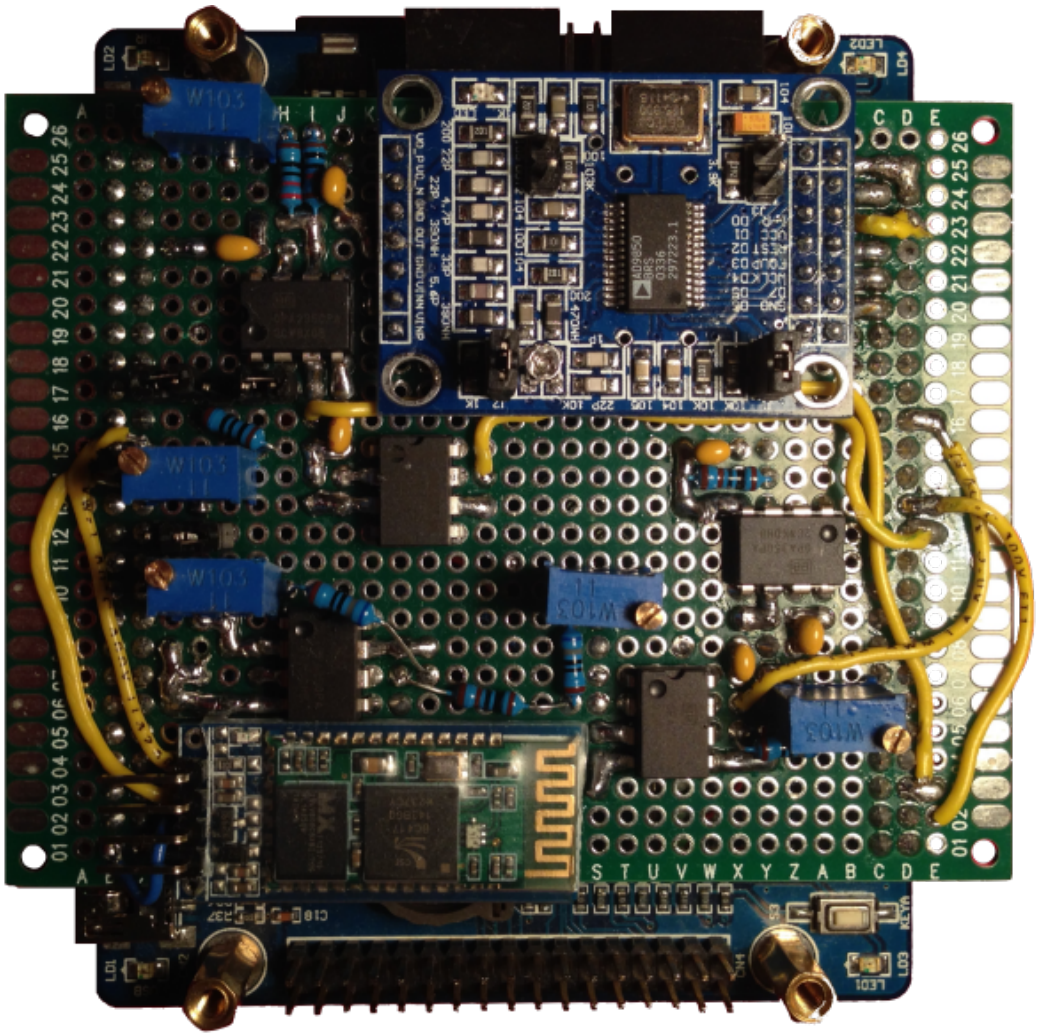


Figure 5.16: A picture of the third prototype.

and renamed "BMSunit" by using "AT+NAMEBMSunit". The default password (called PIN) is "1234" and has not been changed.

It is configured as a slave by default. This means it can not initiate the connection, but accepts incoming connections when the correct password is entered.

The module is shown in figure 5.17.

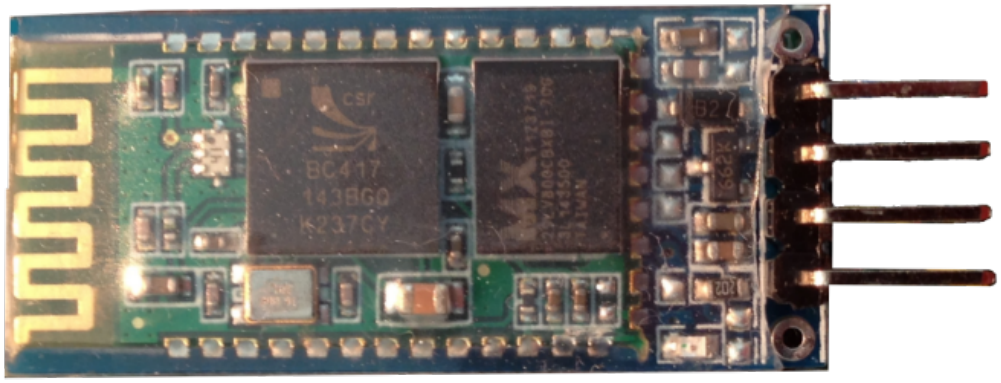



Figure 5.17: A picture of the Bluetooth module.

# Chapter 6

# System Design: Software

## 6.1 Overview

### 6.1.1 Modular code

The microcontroller code have been developed with modularity in mind. Modular programming is a software design technique that apply strict guidelines to how code is organized. Modular code is separated into many files where each file is a part of the whole application with well-defined interfaces. This often reduces the number of dependencies for each file, makes it easier to navigate in the code, maintain the code and is a good foundation for further development.

### 6.1.2 Code abstraction levels

In addition to modularity it is important to reduce the dependencies between different abstraction levels of code. In this thesis the microcontroller code has several levels of abstractions. The lowest abstraction above hardware being the CMSIS, then the Standard Peripheral Library. Above the SPL is the user written drivers, interface and then the application code. The main rule is that each abstraction level can only use and call functions from the layer below. This is shown in figure 6.1.

This improves the code modularity further and if one were to change hardware to a different ARM microcontroller from ST Microelectronics none or minor changes would have to be done in the driver layer. If one were to change to another manufacturer or a different hardware platform the interface and application code would still be working as expected as long as the new hardware platform can facilitate the functionality needed to be able to implement the driver layer.

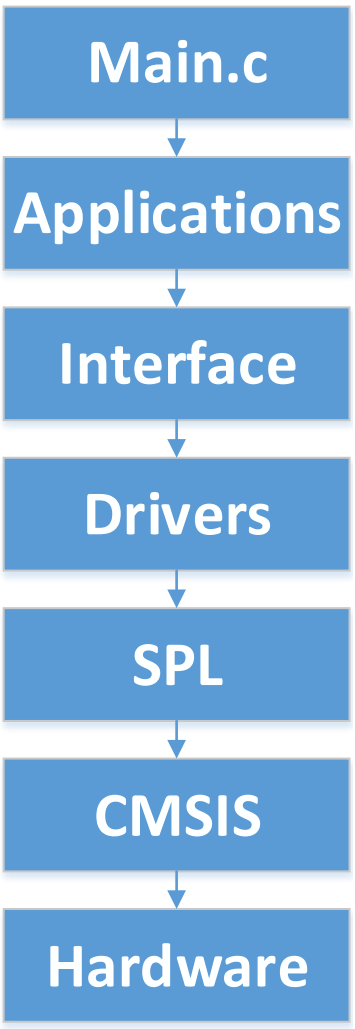



Figure 6.1: Code abstraction levels.

### 6.1.3 Code Organization

The lowest level code is the drivers. This software interacts directly with hardware. In this thesis, the drivers written are shown in table 6.1.

| | | |
|---|---|---|
| ADC | ADC1 | ADC2 |
| ADC3 | DDS | DMA |
| EXTI | FLASH | GPIO |
| IWDG | LED | LSI |
| NVIC | RCC | SYSTICK |
| TIM | TIM1 | TIM2 |
| TIM8 | UART | ZCD |

Table 6.1: List of drivers implemented.

They all use the basic functions in the standard peripheral library (SPL) provided by STMicroelectronics.

Each driver filename is named after the peripheral/hardware they interact with. Therefore, for the analog-to-digital converter the source and header filename is ADC.c and ADC.h respectively. ADC.c/.h only contains functions that are common for all the ADC peripherals. This means that the ADC1.c/.h, ADC2.c/.h and ADC3.c/.h must have common code for it to be moved to the ADC.c/.h. In general, it is desired to move code to the highest layer reasonably possible. This is because changes to code at lower levels larger effect on the rest of the code compared to changes done at higher levels. Simply because code at a lower levels has more code depending on it.

The layer above is called the interface and this code uses the drivers and are in general not allowed to call any code from the SPL or lower. The interfaces implemented are shown in table 6.2.

| | | | |
|---|---|---|---|
| Common | DSP | Initialize | Measurement |
| MessageParser | Queues | Timer | User |

Table 6.2: List of interfaces implemented.

Each interface file is a logical block of the program. For example, the measurement.c/.h contains all functionality related to measurements, such as calibration routines, calculation of measurement parameters, measurement functions and immittance calculations. Another example is the Queues.c/.h containing different queue functionality or Timer.c/.h that has higher-level timing functionality.

The next layer is the application layer. This layer consists of applications using the interface level to perform different tasks. In this thesis, only one

application is running on the system and must handle all states and changes done to it at run-time. The application is called BMS.c/.h. Other applications can be added and an example would be a test application that tests and verifies parts of the code and peripherals functionality.

The microcontroller point of entry is the *main* function in main.c file. This function simply contains an infinite loop (from now on referred to as the main loop) where applications can be run. The main loops responsibility is running the applications in sequential order.

All the code explained above runs in a loop and the code flow is not interrupted and is therefore running as if it was a thread. To be able to asynchronous operations and be able to not use processing power to check if some event has happened, interrupts are used. The interrupts are written in an own file names ISR.c and they will interrupt the programs flow of execution on some event, execute a short routine of code and resume.

The interrupts implemented are shown in table 6.3.

| | | |
|---|---|---|
| DMA1 | DMA2 | EXTI2 |
| SysTick | TIM1 | TIM2 |
| TIM5 | TIM8 | USART1 |

Table 6.3: List of ISRs implemented.

In addition, a header file with widely used constants exists and is named defines.h. This file has all macro definitions and is used by all layers above the SPL.

### 6.1.4 Naming Conventions

This section gives a quick overview of the naming conventions used in the microcontroller C code in this thesis. The rules are listed in tabel 6.4

## 6.2 Microcontroller : Drivers

### 6.2.1 ADC - Analog-to-Digital Converter

The ADC implementation consists of four parts. ADC1, ADC2, ADC3 and the common functions part named ADC.

ADC has functions to enable, disable, enable external triggering and calibrate the three ADCs. A single conversion can be sampled by calling the

| Type | Rule | Example |
|---|---|---|
| Variables and Objects | thisIsAVariableOrStructure | uartRxQueue |
| Function | Filename_FunctionName() | TIM_SetCounterMode() |
| Defines and Enumerations | BIG_LETTERS_ AND_UNDERSCORES | FLASH_BASE_ADDRESS |
| Enumeration and Structure types | typeName_t | ADC_t |
| Boolean variable | isSomethingEnabled | isAdc3Done |

Table 6.4: Naming conventions used in the microcontroller C code in this thesis.

single conversion function. In addition, a function to disable the internal connection to the internal reference voltage and temperature sensor is available.

ADC1 has functions for initializing measurement of the internal reference voltage, 4fs measurement mode and normal lock-in mode.

ADC2 has a function to initialize the ADC2 as a slave for use in dual mode (synchronous mode) with ADC1. This is used for the normal lock-in mode.

ADC3 has an initialize function to ready the ADC3 to measure the excitation voltage on the analog front-end.

### 6.2.2 DMA - Direct Memory Access

The direct memory access has been implemented for ADC1 and ADC3. Two different implementations of the ADC1 exist. One that transfers the converted ADC1 data to a buffer and one that transfers synchronously sampled ADC1 and ADC2 data to a buffer. The DMA for ADC3 is used to transfer the sampled ADC3 data. Two independent DMA controllers exist and DMA2 is used for moving ADC3 data and DMA1 for the two others.

A structure containing the address of the sampling buffer, the number of samples to move and a status flag indicating that the move is done has been made. Get and set functions is available to set and get the values of the structure. Each of the three DMA implementations has its own instance of the structure and an initializer function.

### 6.2.3 DDS - Direct Digital Synthesizer

The DDS has a serial communication interface and a 40-bit register as explained in chapter 5.2.

The DDS can be initialized by calling the configuration function, which sets the frequency, phase offset and if the DDS is to be powered on or off. The current frequency, phase offset and power status of the DDS is obtainable by their respective functions. A structure holds all of the information related to the DDS.

### 6.2.4 EXTI - External Interrupt

All functions for the whole application initializing GPIO as external interrupts are collected in this module.

The ZCD interrupt line is initialized in this module.

### 6.2.5 FLASH - Flash Memory

The flash is used to store data that can be kept even when the system is not powered. On system startup, the stored data is read and determines the system state and settings. This means that if the watchdog resets the system, it might be able to recover without the user noticing.

The initialize function erases all the flash pages and readies them for writing. There are own functions to check if a page is used, convert page to the start address of that page and convert an address to the page number. In addition, a function for erasing all or one page and reading and writing 16-bit, 32-bit, float and double data exists.

A test function testing this module is also available.

### 6.2.6 GPIO - General Purpose Input Output

All GPIO initialization functions for the whole application are collected in this module.

This module also have functions for initializing, setting, clearing and toggling port B pins that are not used elsewhere for debugging purposes. For example to be used with a logic analyzer.

### 6.2.7 IWDG - Independent Watchdog

The independent watchdog (IWDG) is a timer that must be reset within a set time and if it is not, the system will do a software reset. This is done to

make sure the system does not freeze. The watchdog timer uses the LSI as reference and the calibration is necessary to ensure the timeout set is close to the actual timeout. If the timeout is set with an expected LSI clock frequency of 40 kHz when it is in fact 60 kHz, it will timeout two thirds faster than expected and the system might continuously reset itself. An alternative is to set a much higher timeout ensuring that the variations in the LSI clock frequency has less or no effect.

The watchdog can be set up calling the initialize function and then has 200 milliseconds timeout by default. There is a function to set the timeout and start the independent watchdog. The update function is used to keep the system from resetting and a function can be called on system startup to check if the system was reset due to the independent watchdog timeout.

A special function to halt the independent watchdog when debugging is provided and explained why and how to use in chapter 4.3.1.

### 6.2.8 LED - Light Emitting Diode

The light emitting diodes (LED) has been used for simple debugging purposes. A function initializes the LEDs and a different function can turn a specific LED on or off or toggle it.

### 6.2.9 LSI - Low Speed Internal Oscillator

The low speed internal (LSI) RC oscillator has a frequency of 40 kHz. This oscillator is inaccurate and may vary from 30 kHz to 60 kHz. To further enhance its accuracy it can be calibrated. This is done by internally connecting the LSI oscillator to a timer (TIM5 on channel 4). The timer counts with the high speed internal (HSI) oscillator as its reference and is used to accurately calculate the actual frequency of the LSI with the resolution of the HSI. The LSI oscillator is used by other parts of the microcontroller, for example the independent watchdog.

The LSI calibration is done by calling the calibrate function. The frequency is set by TIM5 ISR when calling the calibrate function and can be read by calling the get function afterwards.

### 6.2.10 NVIC - Nested Vector Interrupt Controller

As explained in chapter 4.1.3 the NVIC controls the interrupt priorities.

All NVIC initialization functions for the whole application are collected in this module.

Also a function to be called before using any of the initialize functions is provided. This function sets how many bits to be used for pre-emption priority and sub-priority. Two bits are used for each, which means four levels for each.

### 6.2.11 RCC - Reset and Clock Control

The RCC module controls all the peripherals clocks. Most of the peripherals clock has to be enabled before it can be used. It is done this way to decrease unnecessary power consumption.

All RCC initialization functions for the whole application are collected in this module.

### 6.2.12 SYSTICK - System Tick

The SysTick timer is a 24-bit down counter timer that generates an interrupt when it reaches zero. When zero the SysTick timer is automatically reloaded with the value in the auto-reload register and repeats this process. The Systick timer is a standard timer for all Cortex microcontrollers and can be set up using CMSIS alone. The SysTick is mainly used for providing periodic ticks to a real-time operating system (RTOS) for time keeping or executing periodic tasks.

Calling the initialize function configures the system tick to happen once every millisecond. The delay function can be called to wait for a number of milliseconds in a busy-waiting loop.

### 6.2.13 TIM - Timer

TIM is the timer module. The TIM implementation consists of four parts. TIM1, TIM2, TIM8 and the common functions part named TIM.

TIM has functions to enable and disable timers. A structure containing the timer prescaler, counting mode, clock divider and counting period are found in this module. There are also function to set these values.

TIM1 has an instance of the structure in TIM and an initialize function to configure the TIM1 with the set parameters. In addition, a function to get the TIM1 instance exist.

TIM2 has a function to configure it as a precision hardware timer similar to SYSTICK and a function to get the current tick.

TIM8 has an instance of the structure in TIM and an initialize function to configure the TIM8 with the set parameters. In addition, a function to get the TIM8 instance exist.

### 6.2.14 UART - Universal Asynchronous Receiver/Transmitter

The UART module has an initialize function configuring it for sending and receiving data at 115 200 bits per second. The data length is 1 byte and one stop bit and no parity bit is used. Functions for sending a byte of data or an array is available.

Data is received asynchronously through an ISR or by calling a function that waits until a byte of data is received. Functions for enabling and disabling the ISR has been made. There is also a function for returning a pointer to the UART first-in first-out (FIFO) queue.

### 6.2.15 ZCD - Zero Crossing Detector

The zero crossing detector module has a function to initialize the ZCD. Functions to enable and disable the external interrupt line the ZCD use is available.

## 6.3 Microcontroller : Interface

### 6.3.1 Common

This module contains some common functions not dependent on any of the drivers.

One function converts a value in a known range to the closest equivalent value in an n-bits representation. There is also a function for converting the other way for both signed and unsigned values. These are useful for example to convert the sampled ADC data bits to a correct voltage.

### 6.3.2 DSP - Digital Signal Processing

The DSP module contains a lot functionality. Functions for moving average filter, finding maximum and minimum values, averaging, full wave rectification, digital debouncing filter, FIR filtering and digital lock-in to mention some.

The measurement module heavily uses these functions.

### 6.3.3 Initialize

This module contains a function that collects all initializations to be done at system start-up. It also check whether the system was reset due to the

independent watchdog or just powered on normally.

### 6.3.4 Measurements

The measurement module is one of the largest and most functionality related to measurements, calibrations and immittance calculations are done here.

As an example the Measure_4fs_CSFM() function can be used after some initialization to measure at an input frequency over a chosen number of periods using the 4fs method. After this function has been invoked, all the immittance parameters and IQ demodulation values can be retrieved by calling the appropriate Immittance_Get() function.

System voltage calibration and phase calibration functionality is implemented in this module.

### 6.3.5 Message Parser

The message parser module is responsible for parsing and interpreting the incoming data bytes added to the UART queue by the UART ISR.

The parse new messages function has a 10-millisecond window for receiving UART data and then parse it. When interpreted the commands from the user is set in the application.

### 6.3.6 Queues

Three equal FIFO queues has been implemented for 1, 2 and 4 bytes data length. The implementation is speed optimized and functions for adding data and retrieving next data in queue are the most important. In addition, it is possible to check if the queue is empty, how many elements are currently queued and look at a specified stored element in the queue.

### 6.3.7 Timer

This module has some high level timing functionality. The simplest one being function for delaying of a number of millisecond. There is also a function for starting a timer with a given timeout and a function that can be checked regularly to see if this timeout has happened yet.

### 6.3.8 User

The user module has function to set parameters controlled by the application's user like frequency to measure at, chose the state of the system or do

phase calibration.

## 6.4 Microcontroller : Application

### 6.4.1 BMS - Bioimpedance Measurement System

The BMS application is the highest level of the microcontroller software and are responsible for execution and changing between different modes in the application as startup, idle, measurement and the egg demo.

This application also starts the process of handling incoming messages, update the independent watchdog and sends a periodic heartbeat signal to the connected device. The heartbeat signal is a way of notifying to the connected device that the BMS is running.

## 6.5 Microcontroller : ISR - Interrupt Service Routines

Nine different ISRs has been implemented and are shortly explained here.

EXTI2 interrupt is triggered when the ZCD is enabled and starts the timer TIM1 and disable the ZCD.

DMA1 channel 1 interrupt is used by the 4fs and normal lock-in mode and is triggered when a set of sample data has been transferred by DMA. This ISR disables the timer TIM1, ADC1 and sets the DMA transfer complete flag in the 4fs and normal lock-in mode DMA instances.

DMA2 channel 4 and 5 has the same ISR and is used to notify that the excitation signal has been sampled and data transferred by setting the DMA transfer complete flag in the DMA ADC3 instance.

The SysTick ISR simply decrements the system tick counting values.

Timer TIM1 is used as the sampling timer for ADC1 or ADC1 and ADC2 in dual mode and simply clears the interrupt in the ISR for it to be triggered again.

Timer TIM2 ISR increments a counter used by higher level timing functions.

Timer TIM5 is used to calibrate the low speed internal oscillator and calculated its actual frequency.

Timer TIM8 is identical to timer TIM1, but used as the sampling timer for ADC3.

The USART1 ISR is triggered when incoming UART data is detected and add this data to a queue.

## 6.6 Microcontroller : Other

### 6.6.1 System Event Loop

The Bioimpedance Measurement System (BMS) microcontroller software is run in the systems infinite main loop and is invoked by a single function call. At each iteration of the main loop the commands from the graphical user interface (GUI) are evaluated.

### 6.6.2 Communication

The C standard library function printf() is used to send data as text to the Bluetooth module, which will transfer the data to the connected devices. This is possible by retargeting the printf() function as explained in chapter 4.3.3. Binary data can also be sent byte-wise using the UART_SendData() or UART_SendDataArray() function.

Data received by the Bluetooth module is sent to the microcontroller. The microcontroller is interrupted and an ISR is set up to handle the incoming data. The ISR adds the received byte to a received data queue for later processing.

When enough bytes are received to construct a full message the message is parsed and internal variables are set accordingly. The received messages consists of a preamble, message identifier, payload and a postamble. The preamble is a constant one byte value (0xAA hexadecimal) and signals the start of a message. The message ID is used to tell the system what kind of message it is and which parsing code to use. The payload can vary for the different messages and contains the new information to the system. The message used to control the measurement settings consists of four bytes of binary data. The postamble is similar to the preamble except it is at the end of the message and has a different value (0x0A hexadecimal). The message format for the measurement settings message can be seen in figure 6.2.

### 6.6.3 Measurement Implementation

How the digital lock-in amplifier and 4fs technique use the different hardware, drivers, and algorithms are shown in figure 6.3.

### 6.6.4 4fs method

The signal and reference from the analog front-end (AFE) are connected to the ADC1 and ZCD respectively. The ZCD is used to trigger on the square

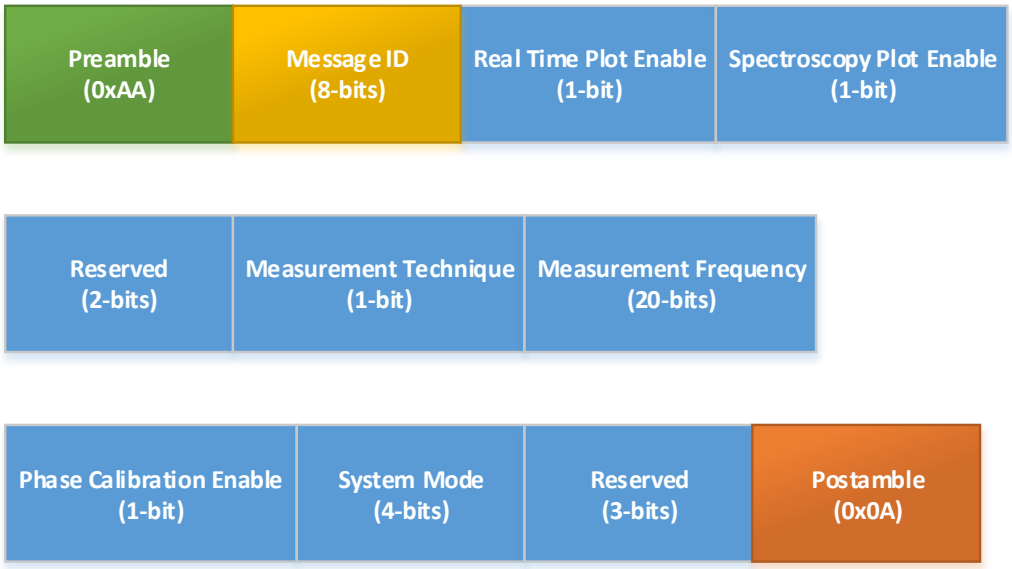



Figure 6.2: Shows the message format of an incoming measurement settings message.

reference signals falling edge to start the sampling timer TIM1. The sampling timer controls the ADC1 that samples the signal from the AFE. The DMA1 automatically move the data from the ADC1 sample data register to an array. When the preset number of samples has been collected and transferred by the DMA1 the 4fs algorithm can use the data.

The 4fs algorithm is shown as a block diagram in figure 6.4. The sampled data is first demodulated using the equations 4.20 and 4.21. Then it is average for some set number of periods and converted to the corresponding voltage values. By knowing the peak-to-peak values of the excitation signal the measured current at the M electrode can be calculated from the voltage value. The next step is calculating all the immittance parameters displayed by the GUI.

### Digital Lock-in

The digital lock-in synchronously sample the signal and reference on ADC1 and ADC2 using the timer TIM1. DMA1 transfers the data to an array for later processing.

The square reference has a digital debouncing filter and is normalized and has either the value 1 or -1. A 90 degrees out of phase square reference is constructed and the signal is multiplied with both the in-phase and quadrature reference. The result is a DC component and a component at twice the

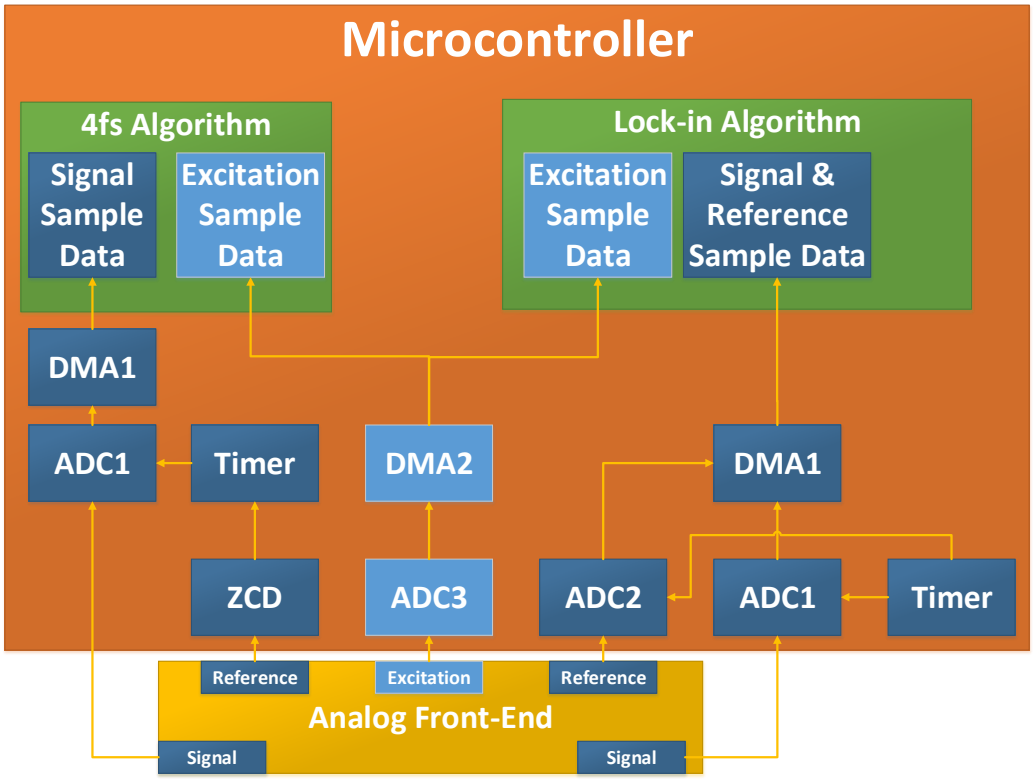



Figure 6.3: An overview of the used hardware and driver modules for the two measurement techniques.

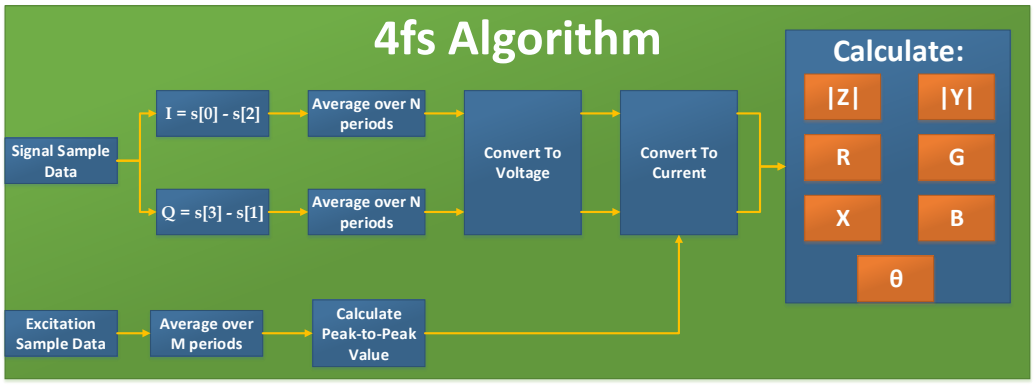



Figure 6.4: A block diagram of the 4fs technique algorithm.

signal frequency as was shown in equation 4.13.

By applying a FIR filter with fixed-point filter coefficients the twice the signal frequency component is removed and the I and Q values are found.

Now the I and Q can be averaged and used to calculate the immittance parameters as done with the 4fs technique.

## 6.7 Graphical User Interface

A graphical user interface (GUI) is supposed to make it easier and more intuitive for a user to interact with an application compared to the command line based applications. It has a big advantage when it comes to visualizing data. An image of the application is shown in figure 6.5.

### 6.7.1 Overview

A GUI has been made to make it easy to control and visualize the measurements by using graphs. The GUI is made with the Qt framework and have been tested on Windows and Android, but might possibly be working on many other platforms such as Linux and Mac.

This chapter will shortly explain the C++ classes of the application.

### 6.7.2 main.cpp

This file is the entry point and contains code that configures the GUI event loop required by the Qt framework, instantiate the application and opens it maximized.

### 6.7.3 Bluetooth

The Bluetooth class is split using two source files for one common header file. One source file is for the Windows version and one for the Android version. By using conditional compilation the appropriate source file is selected. This was necessary as the Qt framework at the time did not support connecting using Bluetooth on Android directly.

The singleton pattern has been used as explained in chapter 4.7.3.

The Windows and the Android version accomplish the same, but are very different when it comes to implementation. The Windows version is implemented using serial port communication as the BMS device will identify as a virtual COM port. This is done in C++ using the QSerialPort class in the Qt framework. On the Android version the Java Native Interface (JNI)

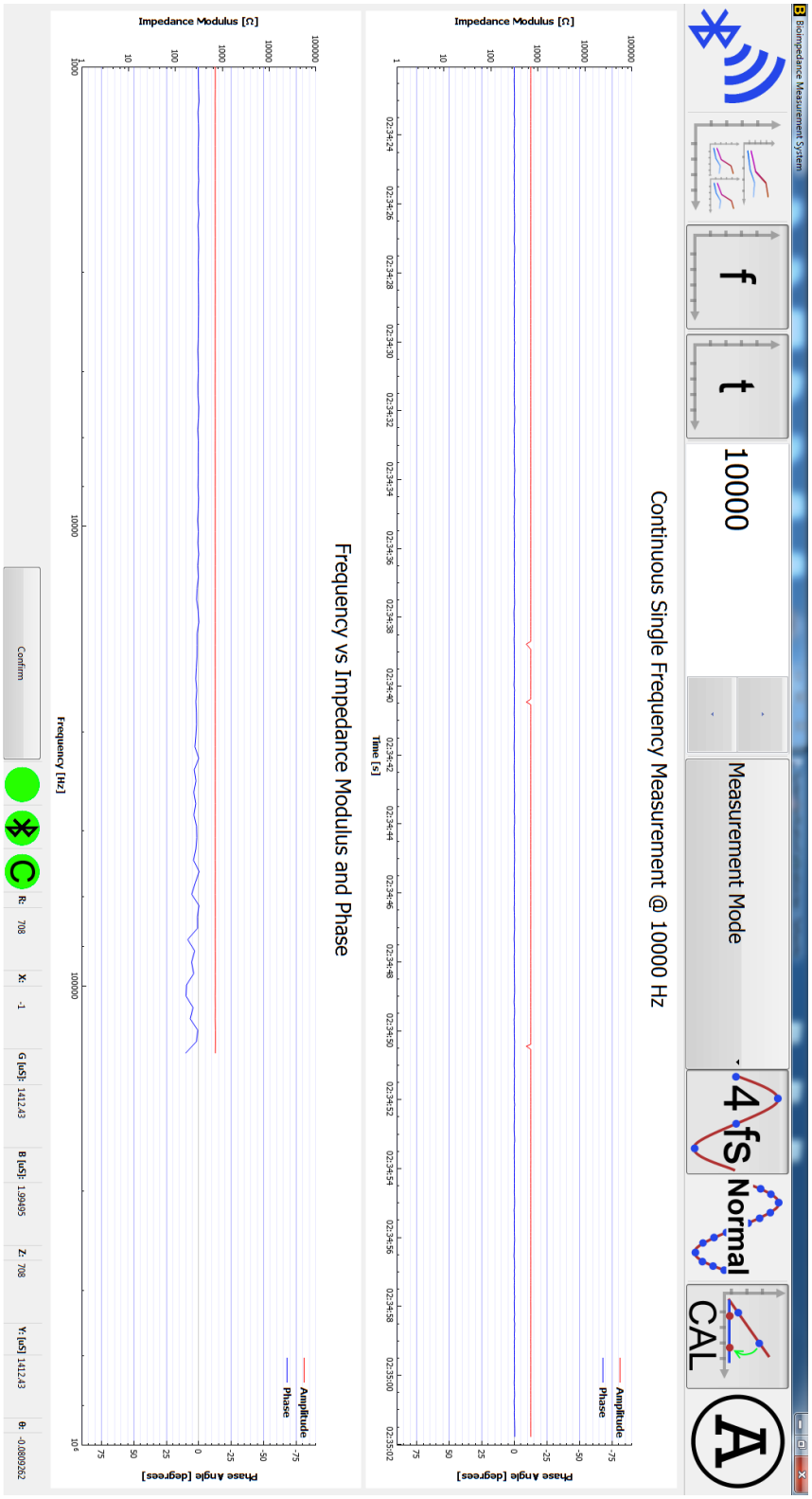



Figure 6.5: An image of the GUI.

had to be utilized to invoke Java code and Android classes in the C++ application. More on how the JNI works is explained in chapter 4.7.2.

The most important implemented methods are the search for devices, connect to device, read and write data methods.

### 6.7.4 Bluetooth Device List

This class inherits the QListWidget class, which provides a basic list with items. This list shows the possible devices or COM ports to connect to. When a device is selected an asynchronous event is sent to initiate the connection.

### 6.7.5 Main Window

This is the top level of the GUI application developed. This class implements the menu bar, tool bar and status bar. The QStackedWidget class has been used to stack the different widgets in the application as the Bluetooth device list and the measurement plots. An instance of the QStackedWidget class is the center visible object in the application and can change it view compared to the static menu bar and tool bar.

In a drop down menu the application modes such as idle, measurement mode and egg demonstrator mode can be selected. This class handles the selection of these modes.

The Main Window class handles all events where the sender and receiver is not within the same class.

### 6.7.6 Measurement Plots

The measurement plots has been implemented using the QCustomPlot library. This class configures a real time plot that plots the incoming phase and impedance modulus data as a function of time. Also a plot showing phase and impedance modulus as a function of frequency has been made. They can both be shown at the same time or just one at a time.

The real time graph is redrawn every 50 millisecond at the most and the other graph every second.

### 6.7.7 Receive

Data received over Bluetooth is sent to this class for parsing. If a valid message has been received the appropriate handler is called and the corresponding data and settings updated.

### 6.7.8 Send

This class readies data before transmission. A preamble, message ID and postamble are added. The messages are also converted to binary data before they are sent with Bluetooth.

# Chapter 7

# System Verification and Calibration

This chapter contains the system verification tests and calibration methods used. All multimeter measurements were done with the Vichy VC99 multimeter.

## 7.1 Testing the DDS

The peak-to-peak voltage amplitude from the DDS was measured with an oscilloscope. The amplitude at the measured frequencies are listed in table 7.1.

| **Frequency** | **DDS peak-to-peak amplitude** |
|---|---|
| 1000 | 1.09 |
| 3000 | 1.13 |
| 5000 | 1.13 |
| 8000 | 1.14 |
| 10000 | 1.13 |
| 50000 | 1.13 |
| 100000 | 1.12 |
| 140000 | 1.12 |

Table 7.1: Table with DDS peak-to-peak amplitude output between 1 kHz and 140 kHz.

## 7.2 Testing the Analog Front-End

A function generator of type AFG320 manufactured by Textronix was used to simulate the DDS input signal to the analog front-end (AFE) and the response on the output was measured with an oscilloscope. The phase difference between the input and the output was measured to be below 10 degrees in the range 1 kHz to 140 kHz.

## 7.3 Microcontroller Interrupts and Events

The behavior and timing of the microcontroller interrupts was measured using a logic analyzer. The logic analyzer used is the Saleae Logic 8 with 24 MHz sampling rate. When testing, each interrupt service routine (ISR) and other events to be measured had a general purpose input/output (GPIO) pin associated with. The respective GPIO was then toggled at the start of the ISR or at events.

In figure 7.1 one can see on channel 6 the square reference signal from the output of the comparator when the DDS output frequency is 10 kHz. When the user sets the desired settings and press the confirm button in the graphical user interface the plot enable on channel 4 is toggled. The zero crossing detector (ZCD) is set to fire only on the falling edge of the reference square signal and at the first occurrence after the plot is enabled, it does. The timer controlling the ADC1's sampling rate is started after the ZCD toggle. The set number of samples are now taken at each toggle of the timer TIM1 on channel 1 in 7.1. When all the samples are collected the direct memory access transfer complete ISR (DMA1 on channel 2) is triggered, signaling that the samples have been moved from the ADC1 to a buffer and is now complete.

The behavior corresponds to what was intended and have been verified. Figure 7.1 is for the 4fs technique and for the normal lock-in mode the only difference would be that the ZCD is not needed to time the start of the sampling and that the timer TIM1 would control the sample time of the ADC1 and ADC2 in dual mode (synchronous sampling).

## 7.4 ADC Calibration

The microcontroller has a built-in self-calibration mode for the analog-to-digital converters (STM32, 2011a, p. 214). The calibration reduces the accuracy errors due to variations in the internal capacitors. After calibration, a digital value has been calculated for each capacitor and this value is now

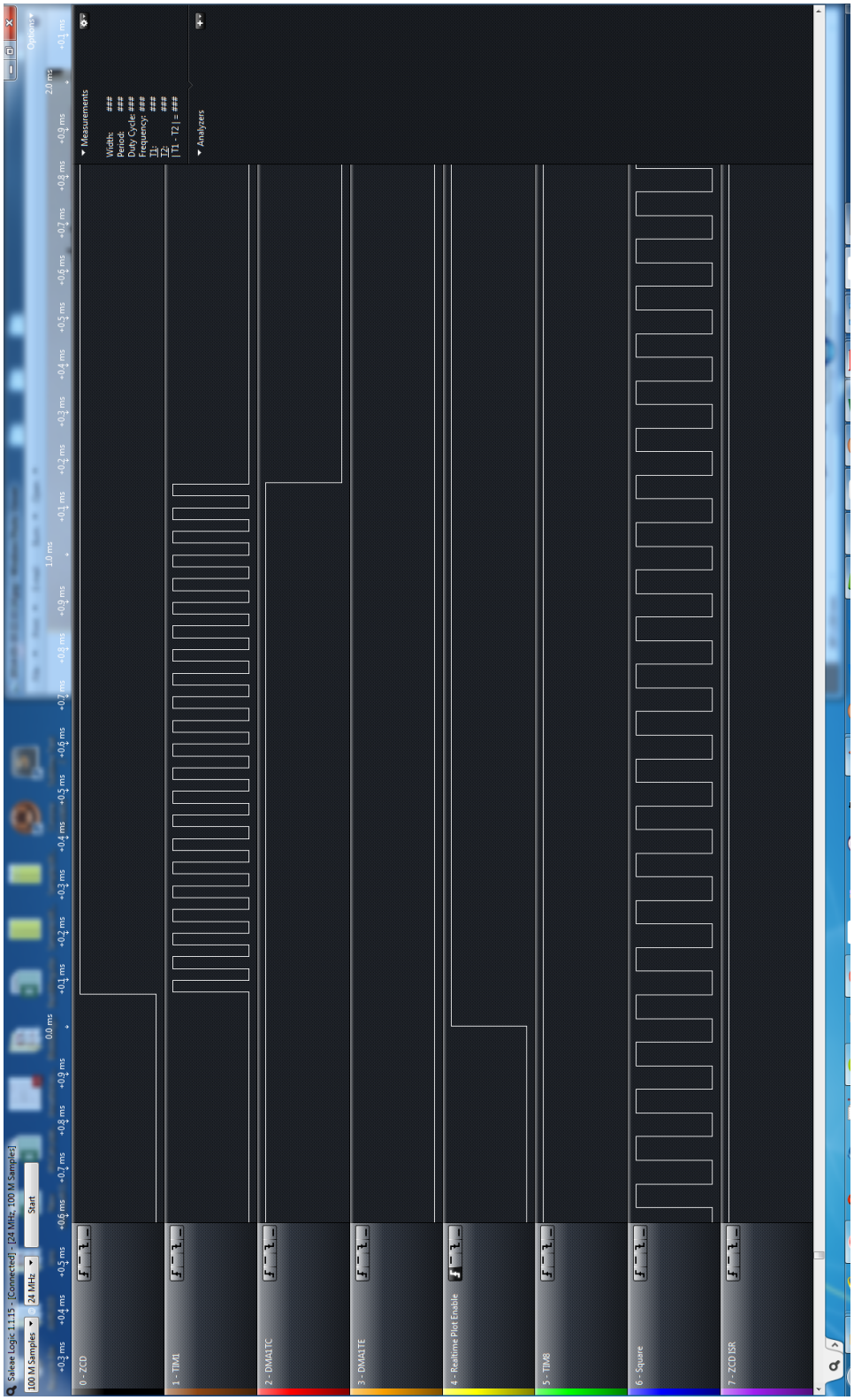



Figure 7.1: A screenshot from the logic analyzer software from Saleae.

automatically subtracted from the converted sampling data. This has been implemented and tested. Three hundred samples have been taken before and after ADC calibration of a constant DC signal and is shown in figure 7.2.

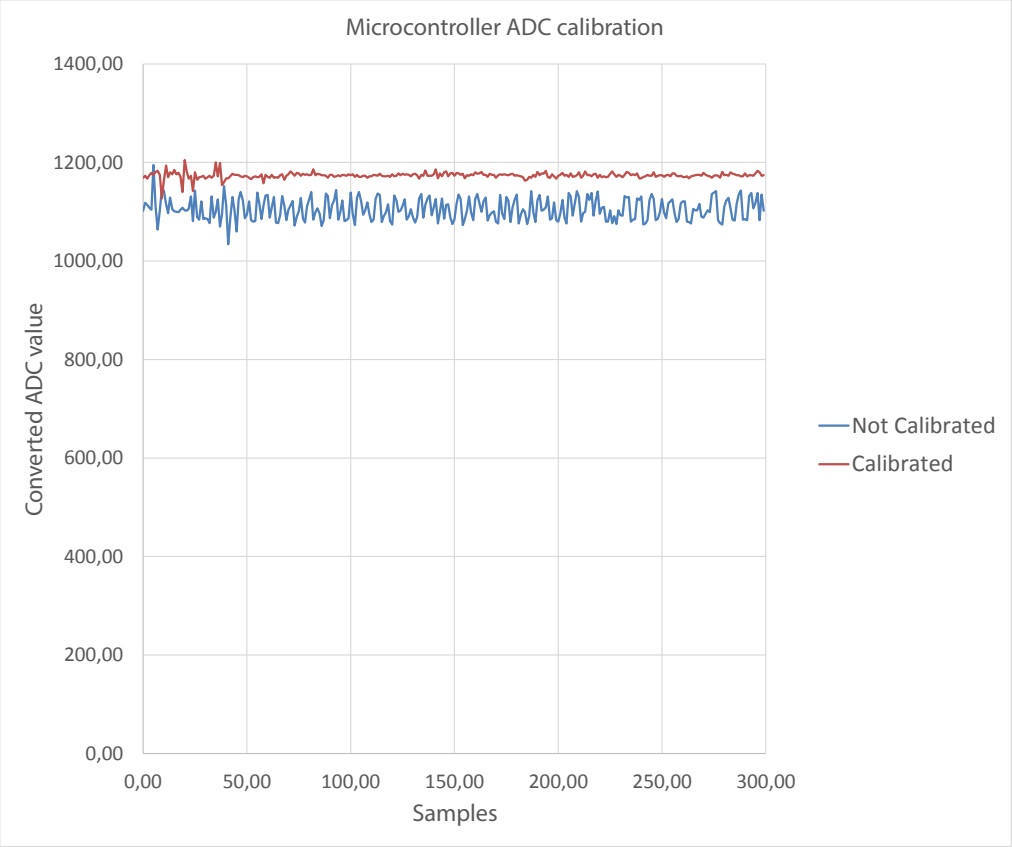



Figure 7.2: Showing the ADC values before and after calibration.

The standard deviation before $\sigma_b$ and after $\sigma_a$ is:

$$\sigma_b = 22.10 \tag{7.1}$$

$$\sigma_a = 6.37 \tag{7.2}$$

This means that 99.73 % ($3\sigma$) of the values with an expected value of $\mu$ are within:

$$\mu \pm 3\sigma_b = \mu \pm 66.30 \tag{7.3}$$

$$\mu \pm 3\sigma_a = \mu \pm 19.11 \tag{7.4}$$

The corresponding voltage value for ±66.30 and ±19.11 when the ADC input range is 3.3 V is:

$$V_{3\sigma_b} = \frac{\pm 3\sigma_b}{\text{ADC resolution} - 1} \cdot V_{ADC} = (\frac{\pm 66.30}{2^{12} - 1}) \cdot 3.3V = \pm 53.4 \text{ mV} \quad (7.5)$$

$$V_{3\sigma_a} = \frac{\pm 3\sigma_a}{\text{ADC resolution} - 1} \cdot V_{ADC} = (\frac{\pm 19.11}{2^{12} - 1}) \cdot 3.3V = \pm 15.7 \text{ mV} \quad (7.6)$$

The $3\sigma$ limit is almost three and a half time closer to the expected value $\mu$ after calibration.

## 7.5 System Voltage Calibration

System voltage calibration means the system is able to accurately measure its own power supply and use this information in measurements. The ADC reference voltage is connected to the same 3.3 V supply as the rest and the system voltage can therefore be used to measure the ADC input range more accurately.

Before a system voltage calibration was used the microcontroller boards 3.3 V supply was measured with a multimeter and the value was hardcoded on to the microcontroller. This will most likely give an offset on the measurements.

Measurements on a variable resistor before and after can be seen in figure 7.3. The x-axis is the true resistance measured with a multimeter and the y-axis is the offset from the true resistance measured with the bioimpedance measurement system.

The calibration was done using the internal reference in the microcontroller. The internal reference $V_{iref}$ is a stable 1.2 V reference voltage with an error of ±0.04 for a temperature range from −40° to 85° Celsius (STM32, 2011b, p. 44). This is used to calculate the system voltage:

$$V_{cc} = \frac{\text{ADC resolution}}{\text{ADC data}} \cdot V_{iref} \quad (7.7)$$

$V_{iref}$ is considered constant and the measured ADC data value is then considered to be the sampled value of $V_{iref}$. The ADC resolution is known and is referenced to the system voltage. A ratio can be calculated and when multiplied with the constant internal reference voltage the system voltage has been found.

The bioimpedance measurement system averages over 50 samples when executing this calibration and this is then the system voltage used.

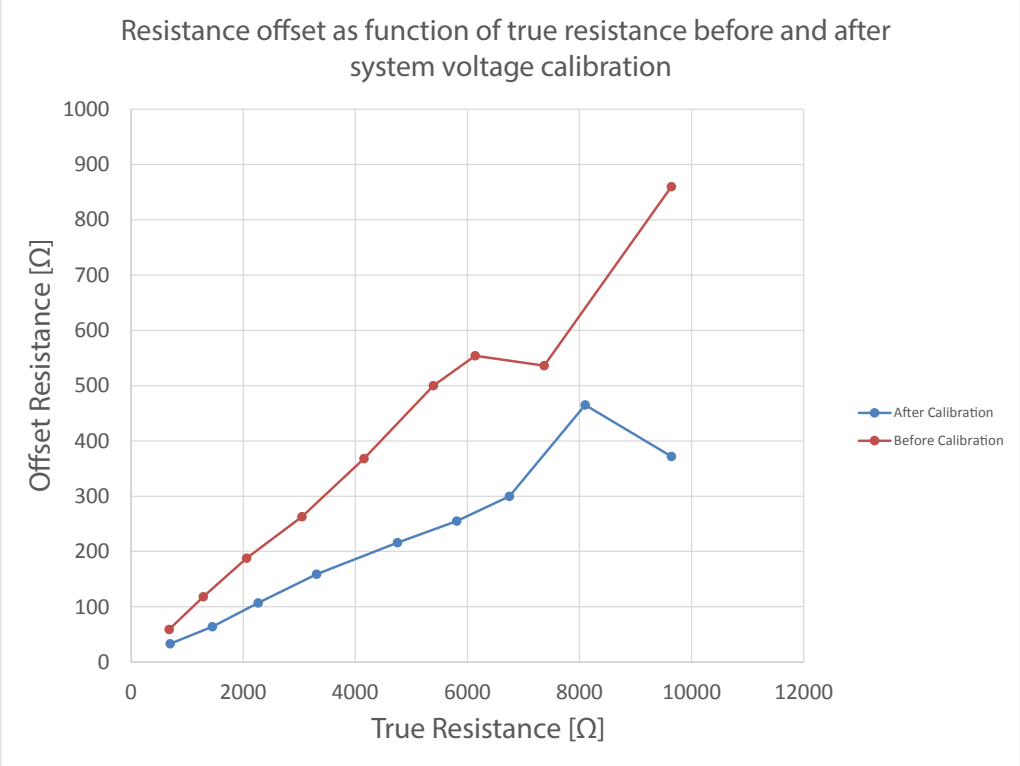



Figure 7.3: Showing the resistance offset before and after system voltage calibration.

## 7.6 Phase Calibration

Three main sources has been identified responsible for phase offset:

1. Comparator delay.
2. Zero crossing detector and firmware delay.
3. Pick-up circuitry delay.

The excitation signal is sent through the comparator and they split their ways, which means the delay of the comparator will delay the reference compared to the signal. The comparator MAX941CPA has a propagation delay of 80 nanoseconds TexasInstruments (2005). By using equation 7.8 the phase delay at the corresponding frequency $f$ can be found.

$$\text{phase delay} = \frac{\text{delay}}{\frac{1}{f}} \cdot 360^\circ \tag{7.8}$$

The comparator phase offset is linear with frequency and corresponds to 0.0288 and 2.88 at 1 kHz and 100 kHz respectively.

The delay between the output of the comparator until the first sample is taken has been measured to be constant for frequencies between 1 kHz and 140 kHz using the logic analyzer. The delay was measured to be $4.000 \pm 0.0417$ $\mu$s. This means equation 7.8 can be used to calculate the delay at the different frequencies. The phase offset due to the zero crossing detector and firmware is $1.44^\circ$ and $144^\circ$ at 1 kHz and 100 kHz respectively.

The circuit from after the M electrode to the output of the second gain stage has a simulated phase delay, which is shown to be linear. The phase offset due to this circuit is approximately $0^\circ$ and $-5^\circ$ at 1 kHz and 100 kHz respectively.

The two first delays can be considered phase delay on the reference signal while the last is phase delay on the excitation signal. All the three delays are linear with frequency.

Because all of the identified delays are linear and the sum of linear equations is a linear equation, a simple two point calibration was chosen. The algorithm implemented starts with measuring the phase at two frequencies, 10 kHz and 100 kHz, and finds the difference:

$$\Delta\theta = \theta_{100\text{kHz}} - \theta_{10\text{kHz}} \tag{7.9}$$

Now the slope can be calculated as:

$$\frac{d\theta}{df} = \frac{\Delta\theta}{100\text{ kHz} - 10\text{ kHz}} \tag{7.10}$$

We can now find the frequency where the phase intersects with the x-axis:

$$\theta_{100\text{kHz}} - (f_{\text{offset}} \cdot \frac{d\theta}{df}) = 0 \tag{7.11}$$

$$\implies f_{\text{offset}} = \frac{\theta_{100\text{kHz}}}{\frac{d\theta}{df}} \tag{7.12}$$

$$f_{\text{zero}} = f_{100kHz} - f_{\text{offset}} \tag{7.13}$$

Here the offset frequency $f_{\text{offset}}$ is the value to subtract from 100 kHz to find the frequency $f_{\text{zero}}$ that is the point where the phase is zero before calibration.

The phase offset to subtract from the measurements at measurement frequency $f$ is then:

$$\theta_{\text{offset}} = (f_{\text{zero}} - f) \cdot \frac{d\theta}{df} \tag{7.14}$$

Figure 7.4 shows the phase before and after the phase calibration on a 678 Ω resistor as function of frequency. The phase calibration must be done on a resistive object for the calibration to be successful, such as a resistor. The implemented calibration on the bioimpedance measurement system averages the phase of 1000 measurements on 10 kHz and 100 kHz.

## 7.7 Measuring: Resistance

Measurements on a variable resistor at 10 kHz is shown as the bottom blue line in figure 7.3. Measurements on a 680 Ω resistor from 1 kHz to 140 kHz is shown on the bottom of figure 7.4.

## 7.8 Measuring: RC series

Measuring on a resistor and capacitor in series was the first test to check the systems phase accuracy. After the system had been phase calibrated, a 10 kΩ variable resistor and a 10 nF capacitor was connected in series on a breadboard. The capacitor was measured to be 9.67 nF with the multimeter. The results between the true resistance of the calculated and measured phase

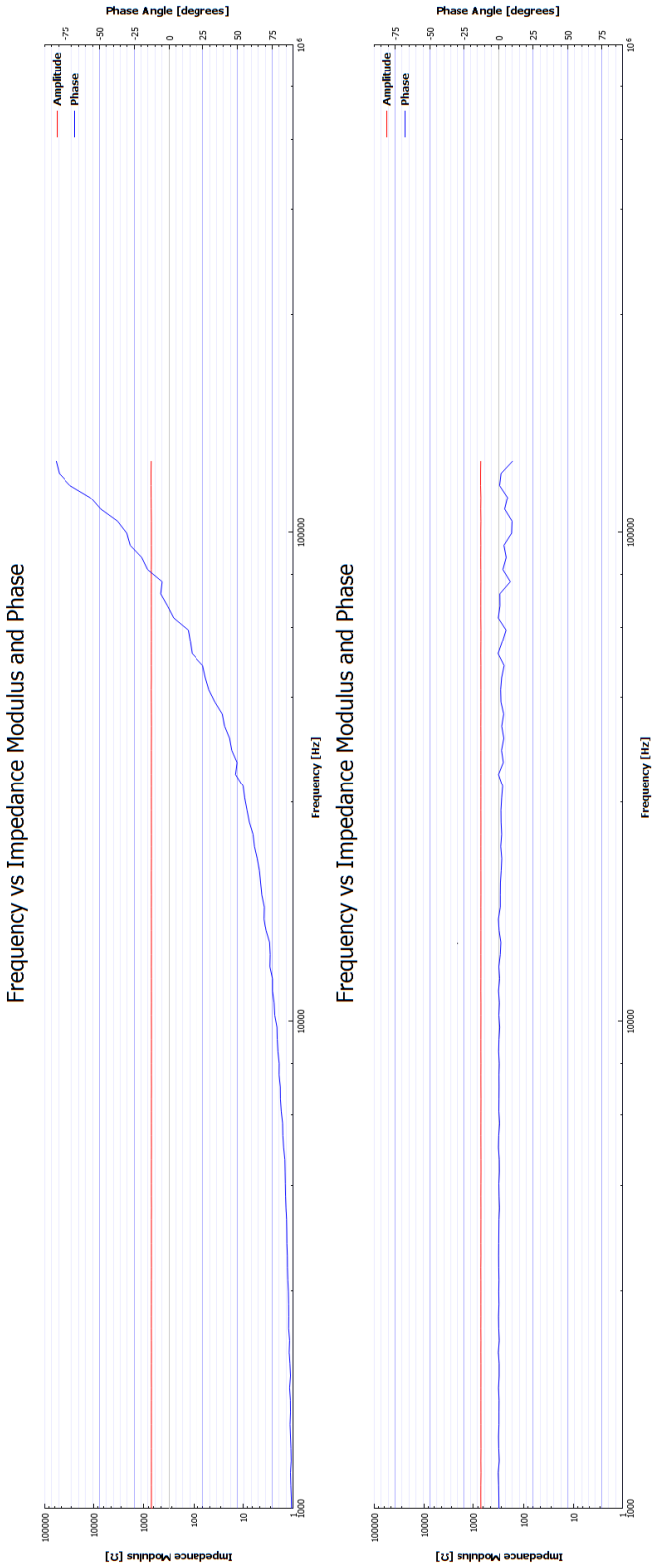



Figure 7.4: Before and after the phase calibration on a resistor.

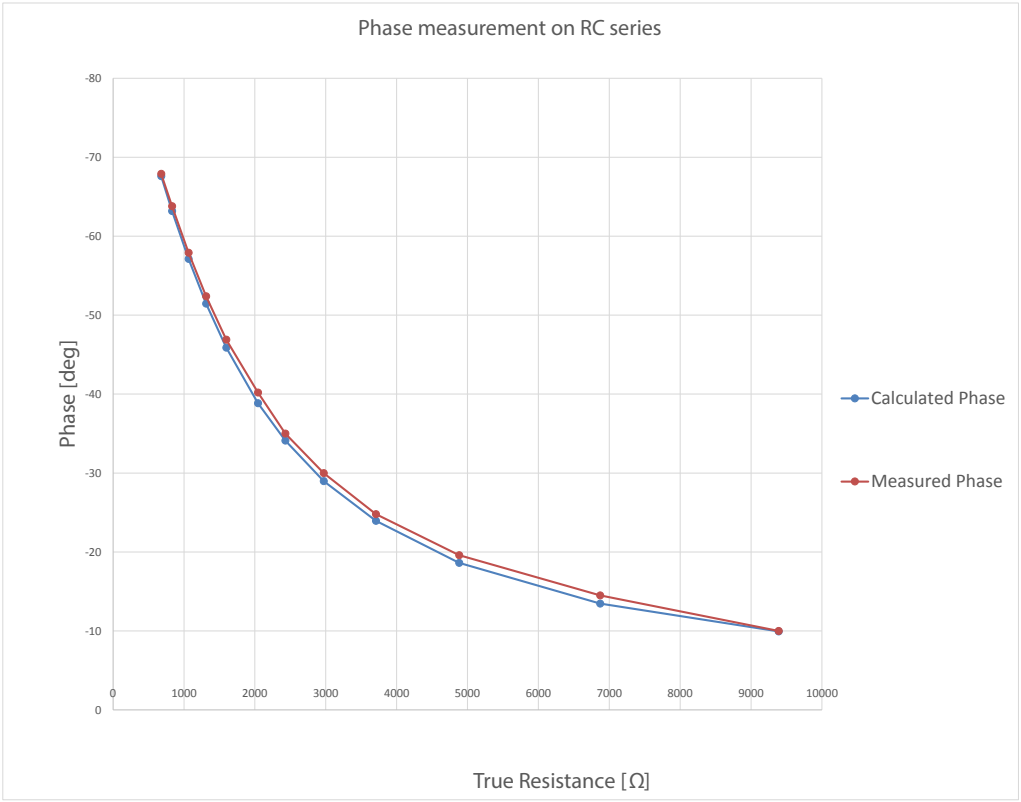



Figure 7.5: The calculated and measured phase as function of resistance in a RC series circuit.

as a function of resistance at 10 kHz is shown in figure 7.5. The measured samples were averaged over 100 periods of the signal.

All the measured values lies within 0.05° - 1.4° above the true calculated values.

## 7.9 Measuring: Boiled Egg

It was desired to measure on some type of biological material and a boiled egg was chosen as the first test because of its clear border between the egg white and egg yolk. Håvard Kalvøy had previous experience measuring on boiled eggs and suggested to discriminate the egg white from the egg yolk by comparing the phases at 3 kHz and 30 kHz. Previous experiments had shown a relative large phase difference between the phases in the egg white compared to the egg yolk.

A two-electrode setup were used. This were realized by connecting the R and C electrode wires together and using a common electrode. The common electrode in this experiment was a metal egg cup. The measuring electrode was a needle electrode manufactured by Medtronic of type "Disposable Monopolar Needle Electrode" with a length of 37 millimeters and a 0.33 millimeter diameter. This needle is insulated except an exposed 0.3 $mm^2$ area on the tip as stated in Kalvoy et al. (2010).

Because of the large size difference between the egg cup electrode and the needle electrode this measurement is considered very monopolar and most of the contributions to the measurement is expected to come from close to the needle tip.

Measurements were done on a well-boiled egg. A saline solution were used to fill the space between the egg and egg cup ensuring good electrical conduction. Three measurements were done on the same egg. The needle were inserted from random positions.

The results are shown in figure 7.6.

It can be seen from the three measurements that the highest phase difference measured for the egg yolk were 22° and the lowest for the egg white were 38°.

The phase limits to use in the algorithm to discriminate the egg white from the egg yolk can now be determined and the limits chosen to be used in the egg demonstrator mode is shown in table 7.2.

In the egg demonstrator mode algorithm is was implemented so that when the impedance modulus is below 100 Ω or above 5 kΩ the needle was considered to not be in the egg.

The egg demonstrator mode and the chosen limits were shown to work

| M1 | \|Z\| Egg yolk | \|Z\| Egg white | Phase Egg yolk | Phase Egg white |
|---|---|---|---|---|
| 3 kHz | 2200 | 1950 | -35 | -54 |
| 30 kHz | 1650 | 850 | -13 | -14 |
| Δ | 550 | 1100 | -22 | -40 |

| M2 | \|Z\| Egg yolk | \|Z\| Egg white | Phase Egg yolk | Phase Egg white |
|---|---|---|---|---|
| 3 kHz | 2200 | 1880 | -28 | -52 |
| 30 kHz | 1460 | 860 | -10 | -12 |
| Δ | 740 | 1020 | -18 | -40 |

| M3 | \|Z\| Egg yolk | \|Z\| Egg white | Phase Egg yolk | Phase Egg white |
|---|---|---|---|---|
| 3 kHz | 2400 | 1920 | -22 | -50 |
| 30 kHz | 1700 | 820 | -9 | -12 |
| Δ | 700 | 1100 | -13 | -38 |

Figure 7.6: The values measured on egg white and egg yolk.

| **Biological material** | **Limit** |
|---|---|
| Egg yolk | Greater than 32° |
| Between egg white and egg yolk | Between 28° and 32° degrees |
| Egg white | Less than 28° degrees |

Table 7.2: Table of egg discimination limits set.

and the egg was sliced so that it could be more accurately verified that the egg yolk was correctly discriminated, but most importantly the border between the egg yolk and egg white. It was not possible by eye measure to see any inaccurate discrimination of the egg white, egg yolk or the boundary between.

## 7.10 Measuring: General Biological Materials

The system is expandable and new algorithms can be added. The impedance modulus and phase can be measured quasi-parallel at several frequencies and any mathematical operations can be used to implement a discrimination algorithm. Two frequencies and the phase difference were used when discriminating the egg white from the egg yolk.

# Chapter 8

# Summary, Conclusion and Future Work

This chapter summarizes the most important parts of the thesis, the results and recommendations for future work.

## 8.1 Conclusion of Present Work

This thesis describes the design, development and verification of a bioimpedance measurement system based on a microcontroller.

The following conclusions are drawn:

- A bioimpedance measurement system prototype for measuring admittance has been developed, tested and verified. The 4fs technique is successfully used for frequencies between 1 kHz and 140 kHz. The normal digital lock-in amplifier code has not been integrated, but the most parts has been individually tested to work, such as the mixing and FIR filtering.
- The system have been made from scratch, making it very flexible and a good platform for further development.
- The analog front-end has been developed, tested and verified to work.
- Wireless communication for transferring data to a computer and control the device from the computer were implemented. The Android software version is currently not correctly transferring data.
- A graphical user interface software with real-time graphing of measurement data has been developed.

- The prototype were able to discriminate the egg white from the egg yolk and accurately determine the boundary between.

## 8.2 Future Work

Based on the work carried out in this thesis these changes and improvements are recommended:

- A natural next step would be a custom analog front-end and microcontroller PCB with surface mount components.
- A virtual ground rail splitter should be used instead of a simple buffered voltage divider.
- The AD9850 DDS chip is not the best DDS considering price and power consumption. The AD9837 is recommended.
- To increase the current range of the transimpedance amplifier, multiple feedback loops with different gains can be added and controlled by the microcontroller. Also a logarithmic transimpedance amplifier can be used.
- A programmable gain amplifier is recommended in the gain stages.
- Further improve the 4fs technique instead of completing the normal digital lock-in amplifier. Unless the goal is to compare these two techniques performance.
- Save data to file for later analysis.

# Bibliography


ARM (2006). *Cortex-M3 Technical Reference Manual, Rev. r1p1*. ARM.

ARM (2010). *Cortex-M3 Devices Generic User Guide*. ARM.

ARM (2012). *Procedure Call Standard for the ARM Architecture*. ARM.

Bhat, A. (2012). Stabilized transimpedance amplifers key to reliable performance. *Maxim Integrated Application Note* (5129).

Cole, K. S. (1940). Permeability and impermeability of cell membranes for ions. *Cold Spring Harbor Symp. Quant. Biol. 8*, 110–122.

Cope, M. K. (2003). Design, simulation and implementation of a digital quadrature demodulator for a stepped frequency radar. *Master Thesis*.

Elwakil, A. S. and B. Maundy (2010). Extracting the cole-cole impedance model parameters without direct impedance measurement. *Electronics Letters 46*(20).

Grimnes, S. and O. G. Martinsen (2006). Bioimpedance. *Wiley Encyclopedia of Biomedical Engineering*.

Grimnes, S. and O. G. Martinsen (2008). *Bioimpedance and Bioelectricity Basics 2nd edition*. Academic Press: San Diego.

Hoyum, P., H. Kalvoy, O. G. Martinsen, and S. Grimnes (2010). A finite element model of needle electrode spatial sensitivity. *Physiological Measurement 31*, 1369–1379.

Ivorra, A. (2003). Bioimpedance monitoring for physicians: an overview. *Centre Nacional de Microelectrònica*.

Kalvoy, H. (2010). Needle guidance in clinical applications based on electrical impedance. *PhD Thesis*.

Kalvoy, H., L. Frich, S. Grimnes, O. G. Martinsen, P. K. Hol, and A. Stubhaug (2009). Impedance-based tissue discrimination for needle guidance. *Physiological Measurement 30*, 129–140.

Kalvoy, H., C. Tronstad, B. Nordbotten, S. Grimnes, and O. G. Martinsen (2010). Electrical impedance of stainless steel needle electrodes. *Annals of Biomedical Engineering 38*(7), 2371–2382.

Keithley Instruments, I. (2004). *Low Level Measurements Handbook, 6th Edition*. Keithley Instruments, Inc.

Martin, T. (2009). *The Insider's Guide To The STM32 ARM Based Microcontroller*. Hitex (UK) Ltd.

Martinsen, O. G. and S. Grimnes (2008). The concept of transfer impedance in bioimpedance measurements. *IFMBE Proceedings 22*, 1079–1079.

Martinsen, O. G., S. Grimnes, and H. P. Schwan (2002). Interface phenomena and dielectric properties of biological tissue. *Encyclopedia of Surface and Colloid Science 8*, 2643–2652.

Nordbotten, B. J. (2008). Bioimpedance measurements using the integrated circuit ad5933. *Master Thesis*.

Orozco, L. (2013). Programmable-gain transimpedance amplifiers maximize dynamic range in spectroscopy systems. *Analog Dialogue 47*(5).

Riu, P. J. (2004). Comments on bioelectrical parameters of the whole human body obtained through bioelectrical impedance analysis. *Bioelectromagnetics 25*, 69–71.

STM32 (2011a). *STM32 Reference Manual (RM0008), Rev. 14*. STM32.

STM32 (2011b). *STM32F103xC, STM32F103xD and STM32F103xE Data Sheet*. STM32.

TexasInstruments (2002). *Op Amps For Everyone*. TexasInstruments.

TexasInstruments (2005). *OPS350 Data Sheet*. TexasInstruments.

TI (2013). Design considerations for a transimpedance amplifier. *Texas Instruments Application Note* (1803).

# Appendix A

# Microcontroller Code

## A.1 Code: C

.

Listing A.1: ADC.h

```
1  #include "stm32f10x.h"
2
3  #ifndef USER_ADC_H_
4  #define USER_ADC_H_
5
6  typedef enum
7  {
8     ADC_1 = 1,
9     ADC_2,
10    ADC_3
11 } ADC_t;
12
13 uint16_t ADC_GetSingleConversion(ADC_t ADC);
14 void ADC_Enable(ADC_t ADC);
15 void ADC_Disable(ADC_t ADC);
16 void ADC_Calibrate(ADC_t ADC);
17 void ADC_ExternalTriggerEnable(ADC_t ADC);
18 void ADC_DisableInternalRef();
19
20 #endif /* USER_ADC_H_ */
```

Listing A.2: ADC.c

```
1  #include "stm32f10x.h"
2  #include "stm32f10x_adc.h"
3  #include "defines.h"
4  #include "ADC.h"
5
6  //Get a single conversion from ADC.
7  uint16_t ADC_GetSingleConversion(ADC_t ADC)
8  {
9     ADC_TypeDef* ADCx;
10    uint16_t data;
11    uint8_t status;
12
13    switch (ADC)
14    {
15       case ADC_1:
16          ADCx = ADC1;
17          break;
18
19       case ADC_2:
20          ADCx = ADC2;
21          break;
22
23       case ADC_3:
24          ADCx = ADC3;
25          break;
26
27       default:
28          status = 1;
29          break;
30    }
31
32    if(status)
33    {
34       DEBUG("Invalid ADC selected to get single conversion");
35       return 0x00;
36    }
37
38    ADC_SoftwareStartConvCmd(ADCx, ENABLE);
39    while( ADC_GetFlagStatus(ADCx, ADC_FLAG_EOC) == RESET);
40    data = ADC_GetConversionValue(ADCx);
41    return data;
42 }
43
44 //Enable ADC.
45 void ADC_Enable(ADC_t ADC)
46 {
47    ADC_TypeDef* ADCx;
48    uint8_t status;
49
50    switch (ADC)
51    {
52       case ADC_1:
53          ADCx = ADC1;
54          break;
55
56       case ADC_2:
57          ADCx = ADC2;
58          break;
59
60       case ADC_3:
```

```
61              ADCx = ADC3;
62              break;
63
64          default:
65              status = 1;
66              break;
67      }
68
69      if(status)
70      {
71          DEBUG("Invalid ADC enabled");
72          return;
73      }
74
75      ADC_Cmd(ADCx, ENABLE);
76  }
77
78  //Disable ADC.
79  void ADC_Disable(ADC_t ADC)
80  {
81      ADC_TypeDef* ADCx;
82      uint8_t status;
83
84      switch (ADC)
85      {
86          case ADC_1:
87              ADCx = ADC1;
88              break;
89
90          case ADC_2:
91              ADCx = ADC2;
92              break;
93
94          case ADC_3:
95              ADCx = ADC3;
96              break;
97
98          default:
99              status = 1;
100             break;
101     }
102
103     if(status)
104     {
105         DEBUG("Invalid ADC disabled");
106         return;
107     }
108
109     ADC_Cmd(ADCx, DISABLE);
110 }
111
112 //Calibrate ADC.
113 void ADC_Calibrate(ADC_t ADC)
114 {
115     ADC_TypeDef* ADCx;
116     uint8_t status;
117
118     switch (ADC)
119     {
120         case ADC_1:
121             ADCx = ADC1;
122             break;
```

```
123
124         case ADC_2:
125             ADCx = ADC2;
126             break;
127
128         case ADC_3:
129             ADCx = ADC3;
130             break;
131
132         default:
133             status = 1;
134             break;
135     }
136
137     if(status)
138     {
139         DEBUG("Invalid ADC selected for calibration");
140         return;
141     }
142
143     //Reset ADCx calibration register.
144     ADC_ResetCalibration(ADCx);
145     //Wait until ADCx reset is done.
146     while(ADC_GetResetCalibrationStatus(ADCx) == SET);
147
148     //Start ADCx calibration.
149     ADC_StartCalibration(ADCx);
150     //Wait until ADCx calibration is done.
151     while(ADC_GetCalibrationStatus(ADCx) == SET);
152 }
153
154 //Enable external triggering of ADC.
155 void ADC_ExternalTriggerEnable(ADC_t ADC)
156 {
157     ADC_TypeDef* ADCx;
158     uint8_t status;
159
160     switch (ADC)
161     {
162         case ADC_1:
163             ADCx = ADC1;
164             break;
165
166         case ADC_2:
167             ADCx = ADC2;
168             break;
169
170         case ADC_3:
171             ADCx = ADC3;
172             break;
173
174         default:
175             status = 1;
176             break;
177     }
178
179     if(status)
180     {
181         DEBUG("Invalid ADC selected for external trigger enable");
182         return;
183     }
184
```

```
185     ADC_ExternalTrigConvCmd(ADCx, ENABLE);
186 }
187 
188 //Disable connection to internal reference voltage and temperature sensor.
189 void ADC_DisableInternalRef()
190 {
191     ADC_TempSensorVrefintCmd(DISABLE);
192 }
```

Listing A.3: ADC1.h

```
1  #include "stm32f10x.h"
2
3  #ifndef USER_ADC1_H_
4  #define USER_ADC1_H_
5
6  void ADC1_4fs_Initialize(void);
7  void ADC1_Normal_Initialize(void);
8  void ADC1_InternalRef_Initialize(void);
9
10 #endif /* USER_ADC1_H_ */
```

Listing A.4: ADC1.c

```
#include "stm32f10x.h"
#include "stm32f10x_adc.h"
#include "ADC.h"
#include "DMA.h"
#include "GPIO.h"
#include "NVIC.h"
#include "RCC.h"

//Initialize ADC1 to be used in 4fs mode with TIM1 as sampling timer and DMA1 to move the data.
void ADC1_4fs_Initialize(void)
{
    ADC_InitTypeDef ADC_InitStructure;

    //Initialize DMA for ADC1 and 4fs mode.
    DMA_ADC1_4fs_Initialize();

    //Configure DMA 1 channel 1 in NVIC.
    NVIC_DMA1_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_0);

    //Configure ADC1 clocks.
    RCC_ADC1_Initialize();

    //Configure ADC1 GPIO.
    GPIO_ADC1_Initialize();

    //Configure ADC 1.
    ADC_Cmd(ADC1, DISABLE);
    ADC_DeInit(ADC1);
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);

    //Register channel 15 on ADC1.
    ADC_RegularChannelConfig(ADC1, ADC_Channel_15, 1, ADC_SampleTime_1Cycles5);

    //Enable DMA on ADC 1.
    ADC_DMACmd(ADC1, ENABLE);

    //Enable ADC 1 to trigger on external interrupts.
    ADC_ExternalTrigConvCmd(ADC1, ENABLE);

    //Enable ADC 1.
    ADC_Enable(ADC_1);

    //Calibrate ADC 1.
    ADC_Calibrate(ADC_1);

    //Disable ADC 1.
    ADC_Disable(ADC_1);
}

//Initialize ADC1 to be used in dual mode with ADC2. To be used for normal lock-in
//mode with TIM1 as sampling timer and DMA1 to move the data.
```

```
58  void ADC1_Normal_Initialize(void)
59  {
60      ADC_InitTypeDef ADC_InitStructure;
61
62      //Configure DMA 1 channel 1 in NVIC.
63      NVIC_DMA1_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_0);
64
65      //Configure ADC1 clocks.
66      RCC_ADC1_Initialize();
67
68      //Configure ADC1 GPIO.
69      GPIO_ADC1_Initialize();
70
71      //Configure ADC 1.
72      ADC_Cmd(ADC1, DISABLE);
73      ADC_DeInit(ADC1);
74      ADC_InitStructure.ADC_Mode = ADC_Mode_RegSimult;
75      ADC_InitStructure.ADC_ScanConvMode = DISABLE;
76      ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
77      ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
78      ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
79      ADC_InitStructure.ADC_NbrOfChannel = 1;
80      ADC_Init(ADC1, &ADC_InitStructure);
81
82      //Register channel 15 on ADC1.
83      ADC_RegularChannelConfig(ADC1, ADC_Channel_15, 1, ADC_SampleTime_7Cycles5
            );
84
85      //Enable DMA on ADC 1.
86      ADC_DMACmd(ADC1, ENABLE);
87
88      //Enable ADC 1 to trigger on external interrupts.
89      ADC_ExternalTrigConvCmd(ADC1, ENABLE);
90
91      //Enable ADC 1.
92      ADC_Enable(ADC_1);
93
94      //Calibrate ADC 1.
95      ADC_Calibrate(ADC_1);
96
97      //Disable ADC 1.
98      ADC_Disable(ADC_1);
99  }
100
101 //Initialize ADC1 to be used to measure on the internal voltage reference.
102 void ADC1_InternalRef_Initialize(void)
103 {
104     ADC_InitTypeDef ADC_InitStructure;
105
106     //Configure ADC1 clocks.
107     RCC_SysVoltage_Initialize();
108
109     //Configure ADC 1.
110     ADC_Cmd(ADC1, DISABLE);
111     ADC_DeInit(ADC1);
112     ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
113     ADC_InitStructure.ADC_ScanConvMode = DISABLE;
114     ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
115     ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
116     ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
117     ADC_InitStructure.ADC_NbrOfChannel = 1;
118     ADC_Init(ADC1, &ADC_InitStructure);
```

```
119
120     //Register channel 17 on ADC1.
121     ADC_RegularChannelConfig(ADC1, ADC_Channel_17, 1,
            ADC_SampleTime_13Cycles5);
122
123     //Enable ADC 1.
124     ADC_Enable(ADC_1);
125
126     //Calibrate ADC 1.
127     ADC_Calibrate(ADC_1);
128
129     //Disable ADC 1.
130     ADC_Disable(ADC_1);
131
132     //Enable the temperature sensor and internal reference channels.
133     ADC_TempSensorVrefintCmd(ENABLE);
134 }
```

Listing A.5: ADC2.h

```
1 #include "stm32f10x.h"
2
3 #ifndef USER_ADC2_H_
4 #define USER_ADC2_H_
5
6 void ADC2_Normal_Initialize(void);
7
8 #endif /* USER_ADC2_H_ */
```

Listing A.6: ADC2.c

```
#include "stm32f10x.h"
#include "stm32f10x_adc.h"
#include "ADC.h"
#include "GPIO.h"
#include "RCC.h"

//Initialize ADC2 to be used as slave with ADC1 in dual mode.
void ADC2_Normal_Initialize(void)
{
   ADC_InitTypeDef ADC_InitStructure;

   //Configure ADC2 clocks.
   RCC_ADC2_Initialize();

   //Configure ADC2 GPIO.
   GPIO_ADC2_Initialize();

   //Configure ADC 2.
   ADC_Cmd(ADC2, DISABLE);
   ADC_DeInit(ADC2);
   ADC_InitStructure.ADC_Mode = ADC_Mode_RegSimult;
   ADC_InitStructure.ADC_ScanConvMode = DISABLE;
   ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
   ADC_InitStructure.ADC_NbrOfChannel = 1;
   ADC_Init(ADC2, &ADC_InitStructure);

   //Register channel 3 on ADC2.
   ADC_RegularChannelConfig(ADC2, ADC_Channel_3, 1, ADC_SampleTime_7Cycles5)
       ;

   //Must enable DMA for the data to be available to be read in the ADC1
       Master DMA data.
   ADC_DMACmd(ADC2, ENABLE);

   //Enable ADC 2 to trigger on external interrupts.
   //Must be enable to be working as slave in Dual ADC mode.
   ADC_ExternalTrigConvCmd(ADC2, ENABLE);

   //Enable ADC 2.
   ADC_Enable(ADC_2);

   //Calibrate ADC 2.
   ADC_Calibrate(ADC_2);

   //Disable ADC 2.
   ADC_Disable(ADC_2);
}
```

Listing A.7: ADC3.h

```
1 #include "stm32f10x.h"
2
3 #ifndef USER_ADC3_H_
4 #define USER_ADC3_H_
5
6 void ADC3_DDS_Initialize(void);
7
8 #endif /* USER_ADC3_H_ */
```

Listing A.8: ADC3.c

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_adc.h"
#include "ADC.h"
#include "GPIO.h"
#include "NVIC.h"
#include "RCC.h"

//Initialize ADC3 to use TIM8 as sampling timer and DMA2 to move the data.
void ADC3_DDS_Initialize(void)
{
    ADC_InitTypeDef ADC_InitStructure;

    //Configure DMA 2 channel 4/5 in NVIC.
    NVIC_DMA2_CH4_5_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);

    //Configure ADC3 clocks.
    RCC_ADC3_Initialize();

    //Configure ADC3 GPIO.
    GPIO_ADC3_Initialize();

    //Configure ADC 3.
    ADC_Cmd(ADC3, DISABLE);
    ADC_DeInit(ADC3);
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T8_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC3, &ADC_InitStructure);

    //Register channel 1 on ADC3.
    ADC_RegularChannelConfig(ADC3, ADC_Channel_1, 1, ADC_SampleTime_7Cycles5)
        ;

    //Enable DMA on ADC 3.
    ADC_DMACmd(ADC3, ENABLE);

    //Enable ADC 3 to trigger on external interrupts.
    ADC_ExternalTrigConvCmd(ADC3, ENABLE);

    //Enable ADC 3.
    ADC_Enable(ADC_3);

    //Calibrate ADC 3.
    ADC_Calibrate(ADC_3);

    //Disable ADC 3.
    ADC_Disable(ADC_3);
}
```

Listing A.9: DDS.h

```
1  #ifndef USER_DDS_H_
2  #define USER_DDS_H_
3
4  #include "defines.h"
5
6  typedef struct DDS
7  {
8     uint32_t frequency;
9     float frequencyFloat;
10    double frequencyDouble;
11    double frequencyError;
12    uint8_t phase;
13    bool_t ddsPowered;
14 } DDS;
15
16 void DDS_Reset(void);
17 uint32_t DDS_Config(uint32_t frequencyWord, uint8_t phase, bool_t powerDown)
       ;
18 uint32_t DDS_GetFrequency(void);
19 float DDS_GetFrequencyFloat(void);
20 double DDS_GetFrequencyDouble(void);
21 double DDS_GetFrequencyError(void);
22 uint8_t DDS_GetPhase(void);
23 bool_t DDS_IsPowered(void);
24
25 #endif /* USER_DDS_H_ */
```

Listing A.10: DDS.c

```
1  #include "stm32f10x.h"
2  #include "stm32f10x_gpio.h"
3  #include "defines.h"
4  #include "DDS.h"
5  #include "GPIO.h"
6  #include "RCC.h"
7
8  static DDS dds;
9
10 //Initialize for communication with the DDS module.
11 static void DDS_Initialize(void)
12 {
13    //Configure DDS clocks.
14    RCC_DDS_Initialize();
15
16    //Configure DDS GPIO.
17    GPIO_DDS_Initialize();
18 }
19
20 //Reset the DDS.
21 void DDS_Reset(void)
22 {
23    //Set control bits to 0.
24    GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
25    GPIO_ResetBits(DDS_PORT_FQUD, DDS_FQUD);
26    GPIO_ResetBits(DDS_PORT_RESET, DDS_RESET);
27
28    //Reset AD9850 registers.
29    GPIO_SetBits(DDS_PORT_RESET, DDS_RESET);
30    GPIO_ResetBits(DDS_PORT_RESET, DDS_RESET);
31
32    //Start and enable serial mode.
33    GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
34    GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
35    GPIO_SetBits(DDS_PORT_FQUD, DDS_FQUD);
36    GPIO_ResetBits(DDS_PORT_FQUD, DDS_FQUD);
37 }
38
39 //Set the frequency, phase offset and to power on/off the DDS.
40 uint32_t DDS_Config(uint32_t freq, uint8_t phase, bool_t powerDown)
41 {
42    uint8_t i;
43    uint32_t ddsClockResolutionMultiple;
44
45    DDS_Initialize();
46
47    //Find corresponding frequencyWord for frequency and write it to the DDS
       frequency word register.
48    dds.frequency = freq;
49    dds.frequencyFloat = DDS_CLOCK_RESOLUTION * ( (uint32_t) (freq /
       DDS_CLOCK_RESOLUTION));
50    dds.frequencyDouble = DDS_CLOCK_RESOLUTION * ( (uint32_t) (freq /
       DDS_CLOCK_RESOLUTION));
51    dds.frequencyError = dds.frequencyFloat - freq;
52
53    //Set the phase offset.
54    dds.phase = phase;
55
56    //Update DDS power status.
57    dds.ddsPowered = powerDown;
```

```
58
59      freq = freq / DDS_CLOCK_RESOLUTION;
60      freq = 0xFFFFFFFF - freq;
61      ddsClockResolutionMultiple = freq;
62
63      DDS_Reset();
64
65      //Load frequency word.
66      for(i = 0; i < 32; i++)
67      {
68          if( freq & (1 << i) )
69          {
70              GPIO_SetBits(DDS_PORT_D7, DDS_D7);
71          }
72          else
73          {
74              GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
75          }
76
77          GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
78          GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
79      }
80
81      //Factory internal test bits (2 bits) must be zero.
82      GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
83      GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
84      GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
85
86      GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
87      GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
88      GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
89
90      //Power down bit.
91      if(powerDown & 0x01)
92      {
93          GPIO_SetBits(DDS_PORT_D7, DDS_D7);
94      }
95      else
96      {
97          GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
98      }
99
100     GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
101     GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
102
103     //Load phase bits.
104     for(i = 0; i < 5; i++)
105     {
106         if( phase & (1 << i) )
107         {
108             GPIO_SetBits(DDS_PORT_D7, DDS_D7);
109         }
110         else
111         {
112             GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
113         }
114
115         GPIO_SetBits(DDS_PORT_WCLK, DDS_WCLK);
116         GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
117     }
118
119     //Load 40-bit serial word.
```

```
120     GPIO_SetBits(DDS_PORT_FQUD, DDS_FQUD);
121     GPIO_ResetBits(DDS_PORT_FQUD, DDS_FQUD);
122
123     //Reset all DDS lines.
124     GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
125     GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
126     GPIO_ResetBits(DDS_PORT_FQUD, DDS_FQUD);
127     GPIO_ResetBits(DDS_PORT_RESET, DDS_RESET);
128
129     //Return DDS_CLOCK_RESOLUTION multiple set in DDS frequency register.
130     return ddsClockResolutionMultiple;
131 }
132
133 //Get the frequency of the DDS.
134 uint32_t DDS_GetFrequency(void)
135 {
136     return dds.frequency;
137 }
138
139 //Get the frequency of the DDS as float.
140 float DDS_GetFrequencyFloat(void)
141 {
142     return dds.frequencyFloat;
143 }
144
145 //Get the frequency of the DDS as double.
146 double DDS_GetFrequencyDouble(void)
147 {
148     return dds.frequencyDouble;
149 }
150
151 //Get the frequency error between set and actual frequency of the DDS.
152 double DDS_GetFrequencyError(void)
153 {
154     return dds.frequencyError;
155 }
156
157 //Get the phase offset of the DDS.
158 uint8_t DDS_GetPhase(void)
159 {
160     return dds.phase;
161 }
162
163 //Check if the DDS is powered.
164 bool_t DDS_IsPowered(void)
165 {
166     return dds.ddsPowered;
167 }
```

Listing A.11: DMA.h

```
#include "stm32f10x.h"
#include "defines.h"

#ifndef USER_DMA_H_
#define USER_DMA_H_

typedef struct DMA
{
   uint32_t samplesAddress;
   uint16_t numberOfSamples;
   volatile bool_t isTransferDone;
} DMA_t;

volatile int isAdc3Done;

//DMA.
DMA_t * DMA_Get4fsStruct(void);
DMA_t * DMA_GetNormalStruct(void);
DMA_t * DMA_GetAdc3Struct(void);
void DMA_SetNumberOfSamples(DMA_t * dma, uint16_t samples);
uint16_t DMA_GetNumberOfSamples(DMA_t * dma);
void DMA_SetSamplingBuffer(DMA_t * dma, int16_t * buffer);
int16_t * DMA_GetSamplingBuffer(DMA_t * dma);
bool_t DMA_SetTransferDone(DMA_t * dma, bool_t value);
bool_t DMA_IsTransferDone(DMA_t * dma);

//DMA_ADC1.
void DMA_ADC1_4fs_Initialize(void);
void DMA_ADC1_Normal_Initialize(void);
void DMA_ADC1_Enable(void);
void DMA_ADC1_Disable(void);

//DMA_ADC3.
void DMA_ADC3_Initialize(void);
void DMA_ADC3_Enable(void);
void DMA_ADC3_Disable(void);

#endif /* USER_DMA_H_ */
```

Listing A.12: DMA.c

```
1  #include "stm32f10x.h"
2  #include "stm32f10x_dma.h"
3  #include "defines.h"
4  #include "DMA.h"
5  #include "RCC.h"
6
7  /* Private structures */
8  static DMA_t dma4fsAdc1;
9  static DMA_t dmaNormalAdc1;
10 static DMA_t dmaAdc3;
11
12 /* DMA */
13 DMA_t * DMA_Get4fsStruct(void)
14 {
15    return &dma4fsAdc1;
16 }
17
18 DMA_t * DMA_GetNormalStruct(void)
19 {
20    return &dmaNormalAdc1;
21 }
22
23 DMA_t * DMA_GetAdc3Struct(void)
24 {
25    return &dmaAdc3;
26 }
27
28 void DMA_SetNumberOfSamples(DMA_t * dma, uint16_t samples)
29 {
30    dma->numberOfSamples = (uint32_t) samples;
31 }
32
33 uint16_t DMA_GetNumberOfSamples(DMA_t * dma)
34 {
35    return dma->numberOfSamples;
36 }
37
38 void DMA_SetSamplingBuffer(DMA_t * dma, int16_t * buffer)
39 {
40    dma->samplesAddress = (uint32_t) &buffer[0];
41 }
42
43 int16_t * DMA_GetSamplingBuffer(DMA_t * dma)
44 {
45    return (int16_t *) dma->samplesAddress;
46 }
47
48 bool_t DMA_SetTransferDone(DMA_t * dma, bool_t value)
49 {
50    dma->isTransferDone = value;
51    return TRUE;
52 }
53
54 bool_t DMA_IsTransferDone(DMA_t * dma)
55 {
56    return dma->isTransferDone;
57 }
58
59 /* DMA_ADC1 */
60 void DMA_ADC1_4fs_Initialize(void)
```

```
61  {
62      DMA_InitTypeDef DMA_InitStructure;
63
64      //Configure DMA1 clocks.
65      RCC_DMA1_Initialize();
66
67      //Configure DMA1 channel 1.
68      DMA_Cmd(DMA1_Channel1, DISABLE);
69      DMA_DeInit(DMA1_Channel1);
70      DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)
            ADC1_DATA_REGISTER_ADDRESS;
71      DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t) dma4fsAdc1.
            samplesAddress;
72      DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
73      DMA_InitStructure.DMA_BufferSize = (uint32_t) dma4fsAdc1.numberOfSamples;
74      DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
75      DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
76      DMA_InitStructure.DMA_PeripheralDataSize =
            DMA_PeripheralDataSize_HalfWord;
77      DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
78      DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
79      DMA_InitStructure.DMA_Priority = DMA_Priority_High;
80      DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
81      DMA_Init(DMA1_Channel1, &DMA_InitStructure);
82
83      //Enable DMA 1 channel 1 Transfer Complete and Transfer Error interrupt.
84      DMA_ITConfig(DMA1_Channel1, (DMA_IT_TC | DMA_IT_TE), ENABLE);
85  }
86
87  void DMA_ADC1_Normal_Initialize(void)
88  {
89      DMA_InitTypeDef DMA_InitStructure;
90
91      //Configure DMA1 clocks.
92      RCC_DMA1_Initialize();
93
94      //Configure DMA1 channel 1.
95      DMA_Cmd(DMA1_Channel1, DISABLE);
96      DMA_DeInit(DMA1_Channel1);
97      DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)
            ADC1_DATA_REGISTER_ADDRESS;
98      DMA_InitStructure.DMA_MemoryBaseAddr = dmaNormalAdc1.samplesAddress;
99      DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
100     DMA_InitStructure.DMA_BufferSize = dmaNormalAdc1.numberOfSamples;
101     DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
102     DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
103     DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
104     DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
105     DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
106     DMA_InitStructure.DMA_Priority = DMA_Priority_High;
107     DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
108     DMA_Init(DMA1_Channel1, &DMA_InitStructure);
109
110     //Enable DMA 1 channel 1 Transfer Complete and Transfer Error interrupt.
111     DMA_ITConfig(DMA1_Channel1, (DMA_IT_TC | DMA_IT_TE), ENABLE);
112 }
113
114 void DMA_ADC1_Enable(void)
115 {
116     //Enable DMA 1 channel 1.
117     DMA_Cmd(DMA1_Channel1, ENABLE);
118 }
```

```
119
120 void DMA_ADC1_Disable(void)
121 {
122    //Disable DMA 1 channel 1.
123    DMA_Cmd(DMA1_Channel1, DISABLE);
124 }
125
126 /* DMA_ADC3 */
127 void DMA_ADC3_Initialize(void)
128 {
129    DMA_InitTypeDef DMA_InitStructure;
130
131    //Configure DMA2 clocks.
132    RCC_DMA2_Initialize();
133
134    //Configure DMA2 channel 5.
135    DMA_Cmd(DMA2_Channel5, DISABLE);
136    DMA_DeInit(DMA2_Channel5);
137    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)
           ADC3_DATA_REGISTER_ADDRESS;
138    DMA_InitStructure.DMA_MemoryBaseAddr = dmaAdc3.samplesAddress;
139    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
140    DMA_InitStructure.DMA_BufferSize = dmaAdc3.numberOfSamples;
141    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
142    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
143    DMA_InitStructure.DMA_PeripheralDataSize =
           DMA_PeripheralDataSize_HalfWord;
144    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
145    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
146    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
147    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
148    DMA_Init(DMA2_Channel5, &DMA_InitStructure);
149
150    //Enable DMA 2 channel 5 Transfer Complete and Transfer Error interrupt.
151    DMA_ITConfig(DMA2_Channel5, (DMA_IT_TC | DMA_IT_TE), ENABLE);
152 }
153
154 void DMA_ADC3_Enable(void)
155 {
156    //Enable DMA 2 channel 5.
157    DMA_Cmd(DMA2_Channel5, ENABLE);
158 }
159
160 void DMA_ADC3_Disable(void)
161 {
162    //Disable DMA 2 channel 5.
163    DMA_Cmd(DMA2_Channel5, DISABLE);
164 }
```

Listing A.13: EXTI.h

```
#ifndef USER_EXTI_H_
#define USER_EXTI_H_

void EXTI_ZDC_Initialization(void);

#endif /* USER_EXTI_H_ */
```

Listing A.14: EXTI.c

```
#include "stm32f10x_exti.h"

//Configure ZCD interrupts on pin 2.
void EXTI_ZDC_Initialization(void)
{
   EXTI_InitTypeDef EXTI_InitStructure;

   //Configure GPIOX pin 2 interrupt.
   EXTI_InitStructure.EXTI_Line = EXTI_Line2;
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
   EXTI_Init(&EXTI_InitStructure);
}
```

Listing A.15: FLASH.h

```
#ifndef USER_FLASH_H_
#define USER_FLASH_H_

#include "defines.h"

//In this enumeration an increment of 1 is 16-bit.
enum {
   FLASH_STATUS_BITS = 0,
   MEASUREMENT_MODE = 1,
   CSFM_FREQUENCY = 2,
   X = 10
} Flash_StoredData;

typedef union {
   double d;
   uint32_t uint[2];
} doubleUint32;

bool_t FLASH_Initialize(void);
bool_t FLASH_isPageUsed(uint8_t page);
uint8_t FLASH_AddressToPage(uint32_t address);
uint32_t FLASH_PageToAddress(uint8_t page);
bool_t Flash_ErasePage(uint8_t page);
uint16_t FLASH_EraseStoragePages(void);
bool_t FLASH_Write16Address(uint32_t address, uint16_t data);
bool_t FLASH_Write32Address(uint32_t address, uint32_t data);
bool_t FLASH_WriteFloatAddress(uint32_t address, float data);
bool_t FLASH_WriteDoubleAddress(uint32_t address, double data);
uint16_t FLASH_Read16(uint32_t address);
uint32_t FLASH_Read32(uint32_t address);
float FLASH_ReadFloat(uint32_t address);
double FLASH_ReadDouble(uint32_t address);
bool_t FLASH_Test_WriteRead(void);

#endif /* USER_FLASH_H_ */
```

Listing A.16: FLASH.c

```
1  #include "Flash.h"
2  #include "stm32f10x.h"
3  #include "stm32f10x_flash.h"
4  #include "defines.h"
5
6  //Erase all flash pages and initialize them for writing.
7  bool_t FLASH_Initialize(void)
8  {
9     uint8_t i;
10    FLASH_Status flashStatus;
11
12    //Unlock flash bank 1.
13    FLASH_UnlockBank1();
14
15    //Clear pending flags.
16    FLASH_ClearFlag(FLASH_FLAG_EOP | FLASH_FLAG_PGERR | FLASH_FLAG_WRPRTERR);
17
18    for(i = 0; i < PAGES_TO_ERASE; i++)
19    {
20       flashStatus = FLASH_ErasePage(FLASH_STORAGE_START_ADDRESS + (
             FLASH_PAGE_SIZE * i));
21       if(flashStatus != FLASH_COMPLETE)
22       {
23          DEBUG("Failed to erase flash page");
24          return FALSE;
25       }
26    }
27    return TRUE;
28 }
29
30 //Checks if a specific page is used.
31 bool_t FLASH_isPageUsed(uint8_t page)
32 {
33    if( (*(uint16_t *)(FLASH_BASE_ADDRESS + (page * FLASH_PAGE_SIZE))) == 0
          xFFFF)
34    {
35       //Page not used.
36       return FALSE;
37    }
38
39    //Page used.
40    return TRUE;
41 }
42
43 //Returns the address to the start of the page of address address, else it
44 //returns 0xFF if outside allowed memory region.
45 uint8_t FLASH_AddressToPage(uint32_t address)
46 {
47    if( (address > (LAST_PAGE_START_ADDRESS + FLASH_PAGE_SIZE)) || (address <
          FLASH_STORAGE_START_ADDRESS) )
48    {
49       //Address is outside the allowed storage memory region.
50       DEBUG("Invalid memory address: %u", (unsigned int) address);
51       return 0xFF;
52    }
53
54    return ((address - FLASH_STORAGE_START_ADDRESS) / FLASH_PAGE_SIZE);
55 }
56
57 //Returns the start address of the page.
```

```
58  uint32_t FLASH_PageToAddress(uint8_t page)
59  {
60     if( (page < 0) || (page > (FLASH_TOTAL_PAGES - 1) ) )
61     {
62        DEBUG("Invalid page selected: %u", page);
63        return 0xFFFFFFFF;
64     }
65
66     return (FLASH_STORAGE_START_ADDRESS + (FLASH_PAGE_SIZE * page));
67  }
68
69  //Erases a specific page.
70  bool_t Flash_ErasePage(uint8_t page)
71  {
72     FLASH_Status flashStatus;
73     uint32_t address;
74
75     address = FLASH_PageToAddress(page);
76
77     flashStatus = FLASH_ErasePage(address);
78     if(flashStatus != FLASH_COMPLETE)
79     {
80        DEBUG("Failed to erase page with address: %u", (unsigned int) (
              FLASH_BASE_ADDRESS + (FLASH_PAGE_SIZE * page)));
81        return FALSE;
82     }
83
84     return TRUE;
85  }
86
87  //Returns 0 is all pages deleted successfully, else it
88  //returns the number of pages that failed to be deleted.
89  uint16_t FLASH_EraseStoragePages(void)
90  {
91     uint8_t i;
92     uint8_t status = 0;
93     FLASH_Status flashStatus;
94
95     for(i = 0; i < ((LAST_PAGE_START_ADDRESS + FLASH_PAGE_SIZE -
           FLASH_STORAGE_START_ADDRESS) / FLASH_PAGE_SIZE); i++)
96     {
97        flashStatus = FLASH_ErasePage(FLASH_STORAGE_START_ADDRESS + (
              FLASH_PAGE_SIZE * i));
98        if(flashStatus != FLASH_COMPLETE)
99        {
100          status++;
101       }
102    }
103
104    if(status != 0)
105    {
106       DEBUG("%u flash pages failed to be erased", status);
107       return status;
108    }
109
110    return 0;
111 }
112
113 //Write 16-bit to an address.
114 bool_t FLASH_Write16Address(uint32_t address, uint16_t data)
115 {
116    FLASH_Status flashStatus;
```

```
117
118 #ifdef DEBUG_MODE
119    //Check if address is aligned. Must be a multiple of 2.
120    if( (address % 2) )
121    {
122       DEBUG("Tried to write to unaligned memory");
123       return FALSE;
124    }
125 #endif
126
127    flashStatus = FLASH_ProgramHalfWord(address, data);
128    if(flashStatus != FLASH_COMPLETE)
129    {
130       DEBUG("Failed to write to flash at address: %u", (unsigned int)
           address);
131       return FALSE;
132    }
133    return TRUE;
134 }
135
136 //Write 32-bit to an address.
137 bool_t FLASH_Write32Address(uint32_t address, uint32_t data)
138 {
139    FLASH_Status flashStatus;
140
141 #ifdef DEBUG_MODE
142    //Check if address is aligned. Must be a multiple of 2.
143    if( (address % 2) )
144    {
145       DEBUG("Tried to write to unaligned memory");
146       return FALSE;
147    }
148 #endif
149
150    flashStatus = FLASH_ProgramWord(address, data);
151    if(flashStatus != FLASH_COMPLETE)
152    {
153       DEBUG("Failed to write to flash at address: %u", (unsigned int)
           address);
154       return FALSE;
155    }
156    return TRUE;
157 }
158
159 //Write a float to an address.
160 bool_t FLASH_WriteFloatAddress(uint32_t address, float data)
161 {
162    FLASH_Status flashStatus;
163
164 #ifdef DEBUG_MODE
165    //Check if address is aligned. Must be a multiple of 2.
166    if( (address % 2) )
167    {
168       DEBUG("Tried to write to unaligned memory");
169       return FALSE;
170    }
171 #endif
172
173    uint32_t * tempPtr = (uint32_t *) &data;
174
175    flashStatus = FLASH_ProgramWord(address, *tempPtr);
176    if(flashStatus != FLASH_COMPLETE)
```

```
177     {
178         DEBUG("Failed to write to flash at address: %u", (unsigned int)
                address);
179         return FALSE;
180     }
181     return TRUE;
182 }
183 
184 //Write a double to an address.
185 //Due to memory alignment of 16-bit, 2 bytes is the finest memory
        granularity. This means when writing double to memory, address % 2 has
        to be 0.
186 bool_t FLASH_WriteDoubleAddress(uint32_t address, double data)
187 {
188     FLASH_Status flashStatus;
189 
190 #ifdef DEBUG_MODE
191     //Check if address is aligned. Must be a multiple of 2.
192     if( (address % 2) )
193     {
194         DEBUG("Tried to write to unaligned memory");
195         return FALSE;
196     }
197 #endif
198 
199     doubleUint32 temp;
200     uint32_t MSB, LSB;
201 
202     temp.d = data;
203 
204     //Get LSB.
205     LSB = temp.uint[0];
206 
207     //Write LSB to flash.
208     flashStatus = FLASH_ProgramWord(address, LSB);
209     if(flashStatus != FLASH_COMPLETE)
210     {
211         DEBUG("Failed to write to flash at address: %u", (unsigned int)
                address);
212         return FALSE;
213     }
214 
215     //Increment address.
216     address += 4;
217 
218     //Get MSB.
219     MSB =  temp.uint[1];
220 
221     //Write MSB to flash.
222     flashStatus = FLASH_ProgramWord(address, MSB);
223     if(flashStatus != FLASH_COMPLETE)
224     {
225         DEBUG("Failed to write to flash at address: %u", (unsigned int)
                address);
226         return FALSE;
227     }
228 
229     return TRUE;
230 }
231 
232 //Reads 16-bits from address.
233 uint16_t FLASH_Read16(uint32_t address)
```

```
234 {
235 #ifdef DEBUG_MODE
236    //Check if address is aligned. Must be a multiple of 2.
237    if( (address % 2) )
238    {
239       DEBUG("Tried to read from unaligned memory");
240       return 0xFFFF;
241    }
242 #endif
243 
244    return (*((uint16_t*) address));
245 }
246 
247 //Reads 32-bits from address.
248 uint32_t FLASH_Read32(uint32_t address)
249 {
250 #ifdef DEBUG_MODE
251    //Check if address is aligned. Must be a multiple of 2.
252    if( (address % 2) )
253    {
254       DEBUG("Tried to read from unaligned memory");
255       return 0xFFFFFFFF;
256    }
257 #endif
258 
259    return (*((uint32_t*) address));
260 }
261 
262 //Reads float from address.
263 float FLASH_ReadFloat(uint32_t address)
264 {
265 #ifdef DEBUG_MODE
266    //Check if address is aligned. Must be a multiple of 2.
267    if( (address % 2) )
268    {
269       DEBUG("Tried to read from unaligned memory");
270       return 0xFFFFFFFF;
271    }
272 #endif
273 
274    return (*((float*) address));
275 }
276 
277 //Reads double from address.
278 //Due to memory alignment of 16-bit, 2 bytes is the finest memory
       granularity. This means reading from memory, address % 2 has to be 0.
279 double FLASH_ReadDouble(uint32_t address)
280 {
281    doubleUint32 temp;
282 
283 #ifdef DEBUG_MODE
284    //Check if address is aligned. Must be a multiple of 2.
285    if( (address % 2) )
286    {
287       DEBUG("Tried to read from unaligned memory");
288       return 0xFFFFFFFFFFFFFFFF;
289    }
290 #endif
291 
292    temp.uint[0] = *((uint32_t *) address);
293    temp.uint[1] = *((uint32_t *) (address + 4));
294 
```

```
295     return temp.d;
296 }
297
298 //A test function for writing and reading all lengths and types to and from
        flash memory.
299 bool_t FLASH_Test_WriteRead(void)
300 {
301     uint8_t status;
302     uint16_t data16;
303     uint32_t data32;
304     double dataDouble;
305     float dataFloat;
306
307     //Erase and initilize all pages in flash memory.
308     status = FLASH_Initialize();
309     if(status)
310     {
311         DEBUG("Flash initialize OK");
312     }
313     else
314     {
315         DEBUG("Flash initialize FAILED");
316         return FALSE;
317     }
318
319     //Write data to memory.
320     FLASH_Write16Address(FLASH_STORAGE_START_ADDRESS+0, 0x1616);
321     FLASH_Write32Address(FLASH_STORAGE_START_ADDRESS+2, 0x32323232);
322     FLASH_WriteFloatAddress(FLASH_STORAGE_START_ADDRESS+6, 21.6125);
323     FLASH_WriteDoubleAddress(FLASH_STORAGE_START_ADDRESS+10, 42.6784864);
324
325     //Read data to memory.
326     data16 = FLASH_Read16(FLASH_STORAGE_START_ADDRESS+0);
327     data32 = FLASH_Read32(FLASH_STORAGE_START_ADDRESS+2);
328     dataFloat = FLASH_ReadFloat(FLASH_STORAGE_START_ADDRESS+6);
329     dataDouble = FLASH_ReadDouble(FLASH_STORAGE_START_ADDRESS+10);
330
331     //Check if read back data is correct.
332     if( data16 != 0x1616 ||
333         data32 != 0x32323232 ||
334         dataFloat != 21.6125 ||
335         dataDouble != 42.6784864)
336     {
337         DEBUG("Flash test failed!");
338         return FALSE;
339     }
340
341     return TRUE;
342 }
```

Listing A.17: GPIO.h

```
#ifndef USER_GPIO_H_
#define USER_GPIO_H_

void GPIO_ADC1_Initialize(void);
void GPIO_ADC2_Initialize(void);
void GPIO_ADC3_Initialize(void);
void GPIO_DDS_Initialize(void);
void GPIO_LED_Initialize(void);
void GPIO_LSI_Initialize(void);
void GPIO_LSI_Deinitialize(void);
void GPIO_USART1_Initialize(void);
void GPIO_ZCD_Initialize(void);

void GPIO_PIN_PORTB_OUT_Initialize(uint16_t pins);
void GPIO_PIN_PORTB_Set(uint16_t pin);
void GPIO_PIN_PORTB_Clear(uint16_t pin);
void GPIO_PIN_PORTB_Toggle(uint16_t pin);

#endif /* USER_GPIO_H_ */
```

Listing A.18: GPIO.c

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "defines.h"

//Initialize ADC1 GPIO.
void GPIO_ADC1_Initialize(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   //Configure PC5 (ADC1 sample input) pin.
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOC, &GPIO_InitStructure);
}

//Initialize ADC2 GPIO.
void GPIO_ADC2_Initialize(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   //Configure PA3 (ADC 2 sample input) pin.
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
}

//Initialize ADC3 GPIO.
void GPIO_ADC3_Initialize(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   //Configure PA1 (ADC 3 sample input) pin.
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
}

//Initialize DDS GPIO.
void GPIO_DDS_Initialize(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   //Configure D7 and WCLK pin.
   GPIO_InitStructure.GPIO_Pin = (DDS_D7 | DDS_WCLK);
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(DDS_PORT_D7, &GPIO_InitStructure);

   //Configure FQUD and RESET pin.
   GPIO_InitStructure.GPIO_Pin = (DDS_FQUD | DDS_RESET);
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(DDS_PORT_FQUD, &GPIO_InitStructure);

   //Reset bits.
```

```
61      GPIO_ResetBits(DDS_PORT_D7, DDS_D7);
62      GPIO_ResetBits(DDS_PORT_WCLK, DDS_WCLK);
63      GPIO_ResetBits(DDS_PORT_FQUD, DDS_FQUD);
64      GPIO_ResetBits(DDS_PORT_RESET, DDS_RESET);
65  }
66
67  //Initialize LED GPIO.
68  void GPIO_LED_Initialize(void)
69  {
70      GPIO_InitTypeDef GPIO_InitStructure;
71
72      //Configure PB0 (LED1) and PB1 (LED2).
73      GPIO_InitStructure.GPIO_Pin = (GPIO_Pin_0 | GPIO_Pin_1);
74      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
75      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
76      GPIO_Init(GPIOB, &GPIO_InitStructure);
77  }
78
79  //Initialize LSI GPIO.
80  void GPIO_LSI_Initialize(void)
81  {
82      //Connect LSI clock output to Timer 5 channel 4 input capture internally.
83      GPIO_PinRemapConfig(GPIO_Remap_TIM5CH4_LSI, ENABLE);
84  }
85
86  //Disable LSI GPIO.
87  void GPIO_LSI_Deinitialize(void)
88  {
89      //Disconnect LSI clock output to Timer 5 channel 4 input capture.
90      GPIO_PinRemapConfig(GPIO_Remap_TIM5CH4_LSI, DISABLE);
91  }
92
93  //Initialize USART1 GPIO.
94  void GPIO_USART1_Initialize(void)
95  {
96      GPIO_InitTypeDef GPIO_InitStructure;
97
98      //Configure PA9 (USART1 Tx).
99      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
100     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
101     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
102     GPIO_Init(GPIOA, &GPIO_InitStructure);
103
104     //Configure PA10 (USART1 Rx).
105     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
106     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
107     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
108     GPIO_Init(GPIOA, &GPIO_InitStructure);
109 }
110
111 //Initialize ZCD GPIO.
112 void GPIO_ZCD_Initialize(void)
113 {
114     GPIO_InitTypeDef GPIO_InitStructure;
115
116     //Configure ZDC pin.
117     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
118     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
119     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
120     GPIO_Init(GPIOE, &GPIO_InitStructure);
121
122     //Selects the GPIO pin used as external interrupt line.
```

```
123     GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource2);
124 }
125
126 //Initialize PORTB pin GPIO.
127 void GPIO_PIN_PORTB_OUT_Initialize(uint16_t pins)
128 {
129     GPIO_InitTypeDef GPIO_InitStructure;
130
131     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
132
133     //Configure GPIO.
134     GPIO_InitStructure.GPIO_Pin = pins;
135     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
136     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
137     GPIO_Init(GPIOB, &GPIO_InitStructure);
138
139     //Clear pins.
140     GPIO_PIN_PORTB_Clear(pins);
141 }
142
143 //Set PORTB pins.
144 void GPIO_PIN_PORTB_Set(uint16_t pin)
145 {
146     GPIOB->BSRR = pin;
147 }
148
149 //Clear PORTB pins.
150 void GPIO_PIN_PORTB_Clear(uint16_t pin)
151 {
152     GPIOB->BRR = pin;
153 }
154
155 //Toggle a single PORTB pin.
156 void GPIO_PIN_PORTB_Toggle(uint16_t pin)
157 {
158     if ((GPIOB->ODR & pin) != (uint32_t)Bit_RESET)
159     {
160         //Bit set, clear it.
161         GPIOB->BRR = pin;
162     }
163     else
164     {
165         //Bit cleared, set it.
166         GPIOB->BSRR = pin;
167     }
168 }
```

Listing A.19: IWDG.h

```
#ifndef USER_WATCHDOG_H_
#define USER_WATCHDOG_H_

void IWDG_Initialize(void);
void IWDG_Start(void);
void IWDG_Update(void);
bool_t IWDG_hasResetHappened(void);
void IWDG_SetTimeout(uint16_t milliseconds);
void IWDG_HaltWhenDebugging(void);



#endif /* USER_WATCHDOG_H_ */
```

Listing A.20: IWDG.c

```
#include "stm32f10x.h"
#include "stm32f10x_iwdg.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"
#include "defines.h"
#include "IWDG.h"
#include "LSI.h"
#include "RCC.h"

//Initializes the independent watchdog with 200 ms timeout.
void IWDG_Initialize(void)
{
   //Configure Low Speed Internal (LSI) clock.
   RCC_IWDG_Initialize();

   //Measure and calibrate LSI oscillator frequency.
   LSI_Calibrate();

   //Enable write access to prescaler and reload registers.
   IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);

   //Set watchdog prescaler.
   IWDG_SetPrescaler(IWDG_Prescaler_4);

   //Disable write access to prescaler and reload registers.
   IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);

   //Set default timeout to 200 ms.
   IWDG_SetTimeout(200);

   //Reset watchdog counter.
   IWDG_Update();
}

//Starts the independent watchdog.
void IWDG_Start(void)
{
   IWDG_Enable();
}

//Updates the independent watchdog.
void IWDG_Update(void)
{
   //Kick the poor dog.
   IWDG_ReloadCounter();
}

//Check at startup if a reset has happened due to watchdog reset.
bool_t IWDG_hasResetHappened(void)
{
   if(RCC_GetFlagStatus(RCC_FLAG_IWDGRST) != RESET)
   {
      //Clear reset flags.
      RCC_ClearFlag();

      return TRUE;
   }
   else
   {
      return FALSE;
```

```
61    }
62 }
63
64 //Set the independent watchdog timeout.
65 //Valid range for prescaler = 4 @ 40 kHz is about 1 to 400 milliseconds.
       (0.1 msec resolution).
66 void IWDG_SetTimeout(uint16_t milliseconds)
67 {
68    uint16_t temp;
69
70    //Check if milliseconds is within range. If not, set to default values.
71    if(milliseconds < 1)
72    {
73       milliseconds = 1;
74    }
75    else if(milliseconds > 400)
76    {
77       milliseconds = 400;
78    }
79
80    //Calculate watchdog reload value.
81    temp = (uint16_t) (((LSI_GetFrequency()/4)*milliseconds)/1000);
82
83    //Enable write access to prescaler and reload registers.
84    IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
85
86    //Set reload value.
87    IWDG_SetReload(temp);
88
89    //Disable write access to prescaler and reload registers.
90    IWDG_WriteAccessCmd(IWDG_WriteAccess_Disable);
91 }
92
93 //Halt the independent watchdog when debugging.
94 void IWDG_HaltWhenDebugging(void)
95 {
96    DBGMCU->CR |= DBGMCU_CR_DBG_IWDG_STOP;
97 }
```

Listing A.21: LED.h

```
#ifndef USER_LED_H_
#define USER_LED_H_

typedef enum
{
   LED1 = 0,
   LED2 = 1
}
LED_t;

typedef enum
{
   OFF = 0,
   ON = 1,
   TOGGLE = 2
} LED_STATUS_t;

void LED_Initialize(void);
void LED(LED_t LED, LED_STATUS_t State);

#endif /* USER_LED_H_ */
```

Listing A.22: LED.c

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "defines.h"
#include "GPIO.h"
#include "LED.h"
#include "RCC.h"

//Initializes the LEDs.
void LED_Initialize(void)
{
   //Configure LED clocks.
   RCC_LED_Initialize();

   //Configure LED GPIO.
   GPIO_LED_Initialize();
}

//Turn on, off or toggle a LED.
void LED(LED_t LED, LED_STATUS_t State)
{
   if (LED == LED1)
   {
      if (State == TOGGLE)
      {
         if (GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_0) == (uint8_t)Bit_SET)
         {
            GPIO_ResetBits(GPIOB, GPIO_Pin_0);
         }
         else
         {
            GPIO_SetBits(GPIOB, GPIO_Pin_0);
         }
      }
      if (State == ON)
      {
         GPIO_SetBits(GPIOB, GPIO_Pin_0);
      }
      if ( State == OFF)
      {
         GPIO_ResetBits(GPIOB, GPIO_Pin_0);
      }
   }
   else
   {
      if (State == TOGGLE)
      {
         if (GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_1) == (uint8_t)Bit_SET)
         {
            GPIO_ResetBits(GPIOB, GPIO_Pin_1);
         }
         else
         {
            GPIO_SetBits(GPIOB, GPIO_Pin_1);
         }
      }
      if (State == ON)
      {
         GPIO_SetBits(GPIOB, GPIO_Pin_1);
      }
      if (State == OFF)
```

```
61          {
62              GPIO_ResetBits(GPIOB, GPIO_Pin_1);
63          }
64      }
65  }
```

Listing A.23: LSI.h

```
#ifndef USER_LSI_H_
#define USER_LSI_H_

void LSI_Calibrate(void);
void LSI_SetCounterValue(uint16_t value);
void LSI_SetFrequency(uint32_t freq);
uint16_t LSI_GetCounterValue(void);
uint32_t LSI_GetFrequency(void);

#endif /* USER_LSI_H_ */
```

Listing A.24: LSI.c

```
1  #include "stm32f10x.h"
2  #include "stm32f10x_tim.h"
3  #include "defines.h"
4  #include "GPIO.h"
5  #include "LSI.h"
6  #include "NVIC.h"
7  #include "RCC.h"
8
9  volatile uint16_t timer5Counter;
10 volatile uint32_t lsiFrequency = 40000;
11
12 //LSI is said to be 40 kHz, but can vary from 30 kHz to 60 kHz. Default
       value is 40 kHz.
13 //LSI is internally connected to timer 5 channel 4. Timer 5 is used with HSE
        crystal to calibrate LSI.
14 void LSI_Calibrate(void)
15 {
16    TIM_ICInitTypeDef TIM_ICInitStructure;
17
18    //Configure TIM5 clock.
19    RCC_LSI_Initialize();
20
21    //Configure TIM5 for use with LSI in NVIC.
22    NVIC_LSI_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);
23
24    //Set Timer 5 prescaler.
25    TIM_PrescalerConfig(TIM5, 0, TIM_PSCReloadMode_Immediate);
26
27    //Connect internal signals.
28    GPIO_LSI_Initialize();
29
30    //Configure Timer 5 as input capture.
31    TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
32    TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
33    TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
34    TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV8;
35    TIM_ICInitStructure.TIM_ICFilter = 0;
36    TIM_ICInit(TIM5, &TIM_ICInitStructure);
37
38    //Reset Timer 5 counter.
39    LSI_SetCounterValue(0);
40
41    //Clear all interrupt flags.
42    TIM5->SR = 0;
43
44    //Enable channel 4.
45    TIM_CCxCmd(TIM5, TIM_Channel_4, TIM_CCx_Enable);
46
47    //Enable Timer 5 capture compare channel 4 interrupt.
48    TIM_ITConfig(TIM5, TIM_IT_CC4, ENABLE);
49
50    //Reset Timer 5.
51    TIM_SetCounter(TIM5, 0);
52
53    //Enable global interrupt.
54    __enable_irq();
55
56    //Enable Timer 5.
57    TIM_Cmd(TIM5, ENABLE);
58
```

```
59      //Wait for LSI calibration to finish.
60      while(LSI_GetCounterValue() < 2);
61
62      //Disable Timer 5 capture compare channel 4 interrupt.
63      TIM_ITConfig(TIM5, TIM_IT_CC4, DISABLE);
64
65      //Disable global interrupt.
66      __disable_irq();
67
68      //Disconnect internal signals.
69      GPIO_LSI_Deinitialize();
70
71      //Reset internal variable.
72      LSI_SetCounterValue(0);
73  }
74
75  void LSI_SetCounterValue(uint16_t value)
76  {
77      timer5Counter = value;
78  }
79
80  void LSI_SetFrequency(uint32_t freq)
81  {
82      lsiFrequency = freq;
83  }
84
85  uint16_t LSI_GetCounterValue(void)
86  {
87      return timer5Counter;
88  }
89
90  uint32_t LSI_GetFrequency(void)
91  {
92      return lsiFrequency;
93  }
```

Listing A.25: NVIC.h

```
1  #include "stm32f10x.h"
2
3  #ifndef USER_NVIC_H_
4  #define USER_NVIC_H_
5
6  typedef enum {
7     NVIC_PRIORITY_0 = 0,
8     NVIC_PRIORITY_1,
9     NVIC_PRIORITY_2,
10    NVIC_PRIORITY_3
11 } NVIC_PRIORITY_t;
12
13 typedef enum {
14    NVIC_SUB_PRIORITY_0 = 0,
15    NVIC_SUB_PRIORITY_1,
16    NVIC_SUB_PRIORITY_2,
17    NVIC_SUB_PRIORITY_3
18 } NVIC_SUB_PRIORITY_t;
19
20 void NVIC_Initialize(void);
21 void NVIC_DMA1_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
22 void NVIC_DMA2_CH4_5_Initialize(NVIC_PRIORITY_t priority,
       NVIC_SUB_PRIORITY_t subPriority);
23 void NVIC_LSI_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
24 void NVIC_SYSTICK_Initialize(void);
25 void NVIC_TIM1_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
26 void NVIC_TIM2_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
27 void NVIC_TIM8_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
28 void NVIC_USART1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
29 void NVIC_ZCD_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority);
30
31 #endif /* USER_NVIC_H_ */
```

Listing A.26: NVIC.c

```
1  #include "stm32f10x.h"
2  #include "NVIC.h"
3  #include "misc.h"
4
5  //Initialize the NVIC priority levels.
6  void NVIC_Initialize(void)
7  {
8     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
9  }
10
11 //priority is pre-emption priority. It can have the value 0-3 where 0 is
       highest.
12 //subPriority is sub-priority. It can have the value 0-3 where 0 is highest.
13
14 //Initialize DMA 1 channel 1 interrupt.
15 void NVIC_DMA1_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority)
16 {
17    NVIC_InitTypeDef NVIC_InitStructure;
18
19    //Register with Nested Vectored Interrupt Controller (NVIC).
20    NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;
21    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
22    NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
23    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
24
25    //Add DMA1_Channel1_IRQn to NVIC.
26    NVIC_Init(&NVIC_InitStructure);
27 }
28
29 //Initialize DMA 2 channel 4/5 interrupt.
30 void NVIC_DMA2_CH4_5_Initialize(NVIC_PRIORITY_t priority,
       NVIC_SUB_PRIORITY_t subPriority)
31 {
32    NVIC_InitTypeDef NVIC_InitStructure;
33
34    //Register with Nested Vectored Interrupt Controller (NVIC).
35    NVIC_InitStructure.NVIC_IRQChannel = DMA2_Channel4_5_IRQn;
36    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
37    NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
38    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
39
40    //Add DMA2_Channel4_5_IRQn to NVIC.
41    NVIC_Init(&NVIC_InitStructure);
42 }
43
44 //Initialize LSI interrupt.
45 void NVIC_LSI_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
       subPriority)
46 {
47    NVIC_InitTypeDef NVIC_InitStructure;
48
49    //Register interrupt with Nested Vectored Interrupt Controller (NVIC).
50    NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
51    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
52    NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
53    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
54
55    //Add TIM5_IRQn to NVIC.
56    NVIC_Init(&NVIC_InitStructure);
```

```
57  }
58
59  //Initialize SysTick interrupt.
60  void NVIC_SYSTICK_Initialize(void)
61  {
62      NVIC_SetPriority(SysTick_IRQn, 0x00);
63  }
64
65  //Initialize TIM 1 channel 1 interrupt.
66  void NVIC_TIM1_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
        subPriority)
67  {
68      NVIC_InitTypeDef NVIC_InitStructure;
69
70      //Register with Nested Vectored Interrupt Controller (NVIC).
71      NVIC_InitStructure.NVIC_IRQChannel = TIM1_CC_IRQn;
72      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
73      NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
74      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
75
76      //Add TIM1_CC_IRQn to NVIC.
77      NVIC_Init(&NVIC_InitStructure);
78  }
79
80  //Initialize TIM 2 channel 1 interrupt.
81  void NVIC_TIM2_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
        subPriority)
82  {
83      NVIC_InitTypeDef NVIC_InitStructure;
84
85      //Register with Nested Vectored Interrupt Controller (NVIC).
86      NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
87      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
88      NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
89      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
90
91      //Add TIM2_IRQn to NVIC.
92      NVIC_Init(&NVIC_InitStructure);
93  }
94
95  //Initialize TIM 8 channel 1 interrupt.
96  void NVIC_TIM8_CH1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
        subPriority)
97  {
98      NVIC_InitTypeDef NVIC_InitStructure;
99
100     //Register with Nested Vectored Interrupt Controller (NVIC).
101     NVIC_InitStructure.NVIC_IRQChannel = TIM8_CC_IRQn;
102     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
103     NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
104     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
105
106     //Add TIM8_CC_IRQn to NVIC.
107     NVIC_Init(&NVIC_InitStructure);
108 }
109
110 //Initialize USART 1 interrupt.
111 void NVIC_USART1_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
        subPriority)
112 {
113     NVIC_InitTypeDef NVIC_InitStructure;
114
```

```
115     //Register with Nested Vectored Interrupt Controller (NVIC).
116     NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
117     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
118     NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
119     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
120
121     //Add USART1_IRQn to NVIC.
122     NVIC_Init(&NVIC_InitStructure);
123 }
124
125 //Initialize ZCD interrupt.
126 void NVIC_ZCD_Initialize(NVIC_PRIORITY_t priority, NVIC_SUB_PRIORITY_t
        subPriority)
127 {
128     NVIC_InitTypeDef NVIC_InitStructure;
129
130     //Nested Vectored Interrupt Controller (NVIC).
131     NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
132     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = priority;
133     NVIC_InitStructure.NVIC_IRQChannelSubPriority = subPriority;
134     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
135
136     //Add EXTI2_IRQn to NVIC.
137     NVIC_Init(&NVIC_InitStructure);
138 }
```

Listing A.27: RCC.h

```
#ifndef USER_RCC_H_
#define USER_RCC_H_

void RCC_ADC1_Initialize(void);
void RCC_ADC2_Initialize(void);
void RCC_ADC3_Initialize(void);
void RCC_DDS_Initialize(void);
void RCC_DMA1_Initialize(void);
void RCC_DMA2_Initialize(void);
void RCC_IWDG_Initialize(void);
void RCC_LED_Initialize(void);
void RCC_LSI_Initialize(void);
void RCC_SysVoltage_Initialize(void);
void RCC_TIM1_Initialize(void);
void RCC_TIM2_Initialize(void);
void RCC_TIM8_Initialize(void);
void RCC_USART1_Initialize(void);
void RCC_ZCD_Initialize(void);

#endif /* USER_RCC_H_ */
```

Listing A.28: RCC.c

```
1  #include "stm32f10x_rcc.h"
2
3  //Initialize ADC1 clock.
4  void RCC_ADC1_Initialize(void)
5  {
6     //Configure ADC clock divider.
7     //ADC clock can be maximum 14 MHz.
8     //PCLK2 = 72 MHz.
9     //Divider can be: 2, 4, 6 or 8.
10    //Fastest ADC clock is:
11    //ADC clock = PCLK2 / 6 = 12 MHz.
12    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
13
14    //Enable ADC1 and GPIO C clock.
15    RCC_APB2PeriphClockCmd( (RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOC) ,
          ENABLE);
16 }
17
18 //Initialize ADC2 clock.
19 void RCC_ADC2_Initialize(void)
20 {
21    //Configure ADC clock divider.
22    //ADC clock can be maximum 14 MHz.
23    //PCLK2 = 72 MHz.
24    //Divider can be: 2, 4, 6 or 8.
25    //Fastest ADC clock is:
26    //ADC clock = PCLK2 / 6 = 12 MHz.
27    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
28
29    //Enable ADC2 and GPIOA clock.
30    RCC_APB2PeriphClockCmd( (RCC_APB2Periph_ADC2 | RCC_APB2Periph_GPIOA) ,
          ENABLE);
31 }
32
33 //Initialize ADC3 clock.
34 void RCC_ADC3_Initialize(void)
35 {
36    //Configure ADC clock divider.
37    //ADC clock can be maximum 14 MHz.
38    //PCLK2 = 72 MHz.
39    //Divider can be: 2, 4, 6 or 8.
40    //Fastest ADC clock is:
41    //ADC clock = PCLK2 / 6 = 12 MHz.
42    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
43
44    //Enable ADC3 and GPIOA clock.
45    RCC_APB2PeriphClockCmd( (RCC_APB2Periph_ADC3 | RCC_APB2Periph_GPIOA) ,
          ENABLE);
46 }
47 //Initialize DDS clock.
48 void RCC_DDS_Initialize(void)
49 {
50    //Enable GPIOB and GPIOE clock.
51    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
52    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
53 }
54
55 //Initialize DMA1 clock.
56 void RCC_DMA1_Initialize(void)
57 {
```

```
58     //Enable DMA1 clock.
59     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
60  }
61
62  //Initialize DMA2 clock.
63  void RCC_DMA2_Initialize(void)
64  {
65     //Enable DMA 2 clock.
66     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA2, ENABLE);
67  }
68
69  //Initialize IWDG clock.
70  void RCC_IWDG_Initialize(void)
71  {
72     //Enable the Low Speed Internal oscillator.
73     RCC_LSICmd(ENABLE);
74
75     //Wait for LSI to stabilize.
76     while(RCC_GetFlagStatus(RCC_FLAG_LSIRDY) == RESET);
77  }
78
79  //Initialize LED clock.
80  void RCC_LED_Initialize(void)
81  {
82     //Enable GPIOB clock.
83     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
84  }
85
86  //Initialize LSI clock.
87  void RCC_LSI_Initialize(void)
88  {
89     //Enable TIM5 clock.
90     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
91  }
92
93  //Initialize SysVoltage clock.
94  void RCC_SysVoltage_Initialize(void)
95  {
96     //Configure ADC clock divider.
97     //ADC clock can be maximum 14 MHz.
98     //PCLK2 = 72 MHz.
99     //Divider can be: 2, 4, 6 or 8.
100    //Fastest ADC clock is:
101    //ADC clock = PCLK2 / 6 = 12 MHz.
102    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
103
104    //Enable ADC1 and GPIO C clock.
105    RCC_APB2PeriphClockCmd( RCC_APB2Periph_ADC1 , ENABLE);
106 }
107
108 //Initialize TIM1 clock.
109 void RCC_TIM1_Initialize(void)
110 {
111    //Enable TIM1 clock.
112    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
113 }
114
115 //Initialize TIM2 clock.
116 void RCC_TIM2_Initialize(void)
117 {
118    //Enable TIM2 clock.
119    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
```

```
120 }
121
122 //Initialize TIM8 clock.
123 void RCC_TIM8_Initialize(void)
124 {
125    //Enable TIM8 clock.
126    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
127 }
128
129 //Initialize USART1 clock.
130 void RCC_USART1_Initialize(void)
131 {
132    //Enable GPIOA, USART1 and AFIO clock.
133    RCC_APB2PeriphClockCmd((RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
           RCC_APB2Periph_AFIO), ENABLE);
134 }
135
136 //Initialize ZCD clock.
137 void RCC_ZCD_Initialize(void)
138 {
139    //Enable AFIO and GPIOE clock.
140    RCC_APB2PeriphClockCmd((RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO),
           ENABLE);
141 }
```

Listing A.29: SYSTICK.h

```
#ifndef USER_SYSTEMTICK_H_
#define USER_SYSTEMTICK_H_

#include "stdint.h"

extern volatile uint32_t systemTickCounter;

void SYSTICK_Initialize(void);
void SYSTICK_Delay(volatile uint32_t milliseconds);
void SYSTICK_DecrementCounter(void);

#endif /* USER_SYSTEMTICK_H_ */
```

Listing A.30: SYSTICK.c

```c
#include "stm32f10x.h"
#include "defines.h"
#include "SYSTICK.h"
#include "NVIC.h"

volatile uint32_t systemTickCounter = 0;

//Initialize the system tick with 1 ms tick interval.
void SYSTICK_Initialize(void)
{
   //Set system tick to 1 millisecond.
   if(SysTick_Config(SYSTEM_FREQUENCY / 1000))
   {
      //Failed to set System Tick interval.
      while(1)
      {
         //To catch this error when debugging.
      }
   }

   //Set system tick priority.
   NVIC_SYSTICK_Initialize();
}

//This function call will wait milliseconds milliseconds before continuing.
void SYSTICK_Delay(volatile uint32_t milliseconds)
{
   //Copy to volatile global variable so ISR can access it.
   systemTickCounter = milliseconds;

   //Enable System Tick counter.
   SysTick->CTRL |= SysTick_CTRL_ENABLE;

   //Wait until counter reaches zero.
   while(systemTickCounter != 0);

   //Disable System Tick counter.
   SysTick->CTRL &= ~SysTick_CTRL_ENABLE;

   //Clear System Tick counter.
   SysTick->VAL = (uint32_t) 0x0;
}

//Function to decrement the system tick counter.
void SYSTICK_DecrementCounter(void)
{
   //Decrement counter until reaches zero.
   if(systemTickCounter != 0x00)
   {
      systemTickCounter--;
   }
}
```

Listing A.31: TIM.h

```
#ifndef USER_TIM_H_
#define USER_TIM_H_

typedef struct TIMER
{
   uint16_t prescaler;
   uint16_t counterMode;
   uint16_t period;
   uint16_t clockDivider;
} TIMER_t;

typedef enum
{
   TIM_1 = 1,
   TIM_2,
   TIM_3,
   TIM_4,
   TIM_5,
   TIM_6,
   TIM_7,
   TIM_8,
   TIM_9,
   TIM_10,
   TIM_11,
   TIM_12,
   TIM_13,
   TIM_14
} TIM_t;

typedef enum
{
   TIM_CLK_DIV_1 = 0,
   TIM_CLK_DIV_2,
   TIM_CLK_DIV_4
} TIM_CLK_DIV_t;

typedef enum
{
   TIM_CNT_MODE_UP = 0,
   TIM_CNT_MODE_DOWN
} TIM_CNT_MODE_t;

void TIM_Enable(TIM_t timer);
void TIM_Disable(TIM_t timer);
void TIM_SetPrescaler(TIMER_t * tim, uint16_t prescaler);
void TIM_SetCounterMode(TIMER_t * tim, TIM_CNT_MODE_t counterMode);
void TIM_SetPeriod(TIMER_t * tim, uint16_t period);
void TIM_SetClockDivider(TIMER_t * tim, TIM_CLK_DIV_t clockDivider);

#endif /* USER_TIM_H_ */
```

Listing A.32: TIM.c

```
#include "stm32f10x.h"
#include "stm32f10x_tim.h"
#include "defines.h"
#include "TIM.h"

//Enable TIM.
void TIM_Enable(TIM_t timer)
{
   TIM_TypeDef* TIMx;
   uint8_t status = 0;

   switch (timer)
   {
      case TIM_1:
         TIMx = TIM1;
         break;

      case TIM_2:
         TIMx = TIM2;
         break;

      case TIM_3:
         TIMx = TIM3;
         break;

      case TIM_4:
         TIMx = TIM4;
         break;

      case TIM_5:
         TIMx = TIM5;
         break;

      case TIM_6:
         TIMx = TIM6;
         break;

      case TIM_7:
         TIMx = TIM7;
         break;

      case TIM_8:
         TIMx = TIM8;
         break;

      case TIM_9:
         TIMx = TIM9;
         break;

      case TIM_10:
         TIMx = TIM10;
         break;

      case TIM_11:
         TIMx = TIM11;
         break;

      case TIM_12:
         TIMx = TIM12;
         break;
```

```
61
62          case TIM_13:
63              TIMx = TIM13;
64              break;
65
66          case TIM_14:
67              TIMx = TIM14;
68              break;
69
70          default:
71              status = 1;
72              break;
73      }
74
75      if(status)
76      {
77          DEBUG("Invalid timer enabled");
78          return;
79      }
80
81      //Clear timer
82      TIM_SetCounter(TIMx, 0);
83
84      //Enable timer
85      TIM_Cmd(TIMx, ENABLE);
86  }
87
88  //Disable TIM.
89  void TIM_Disable(TIM_t timer)
90  {
91      TIM_TypeDef* TIMx;
92      uint8_t status = 0;
93
94      switch (timer)
95      {
96          case TIM_1:
97              TIMx = TIM1;
98              break;
99
100         case TIM_2:
101             TIMx = TIM2;
102             break;
103
104         case TIM_3:
105             TIMx = TIM3;
106             break;
107
108         case TIM_4:
109             TIMx = TIM4;
110             break;
111
112         case TIM_5:
113             TIMx = TIM5;
114             break;
115
116         case TIM_6:
117             TIMx = TIM6;
118             break;
119
120         case TIM_7:
121             TIMx = TIM7;
122             break;
```

```
123
124         case TIM_8:
125            TIMx = TIM8;
126            break;
127
128         case TIM_9:
129            TIMx = TIM9;
130            break;
131
132         case TIM_10:
133            TIMx = TIM10;
134            break;
135
136         case TIM_11:
137            TIMx = TIM11;
138            break;
139
140         case TIM_12:
141            TIMx = TIM12;
142            break;
143
144         case TIM_13:
145            TIMx = TIM13;
146            break;
147
148         case TIM_14:
149            TIMx = TIM14;
150            break;
151
152         default:
153            status = 1;
154            break;
155      }
156
157      if(status)
158      {
159         DEBUG("Invalid timer disabled");
160         return;
161      }
162
163      //Disable timer
164      TIM_Cmd(TIMx, DISABLE);
165   }
166
167   //Set the prescaler.
168   //This parameter can be a number between 0x0000 and 0xFFFF
169   void TIM_SetPrescaler(TIMER_t * tim, uint16_t prescaler)
170   {
171      tim->prescaler = prescaler;
172   }
173
174   //Set counting mode up or down.
175   void TIM_SetCounterMode(TIMER_t * tim, TIM_CNT_MODE_t counterMode)
176   {
177      switch (counterMode) {
178         case TIM_CNT_MODE_UP:
179            tim->counterMode = TIM_CounterMode_Up;
180            break;
181
182         case TIM_CNT_MODE_DOWN:
183            tim->counterMode = TIM_CounterMode_Down;
184            break;
```

```
185
186         default:
187             DEBUG("Invalid Timer Count Mode selected\n");
188             tim->counterMode = (uint16_t) TIM_CounterMode_Up;
189             break;
190     }
191 }
192
193 //Set the counting period.
194 //This parameter can be a number between 0x0000 and 0xFFFF
195 void TIM_SetPeriod(TIMER_t * tim, uint16_t period)
196 {
197     tim->period = period;
198 }
199
200
201 //Set the clock divider.
202 void TIM_SetClockDivider(TIMER_t * tim, TIM_CLK_DIV_t clockDivider)
203 {
204     switch (clockDivider) {
205         case TIM_CLK_DIV_1:
206             tim->clockDivider = TIM_CKD_DIV1;
207             break;
208
209         case TIM_CLK_DIV_2:
210             tim->clockDivider = TIM_CKD_DIV2;
211             break;
212
213         case TIM_CLK_DIV_4:
214             tim->clockDivider = TIM_CKD_DIV4;
215             break;
216
217         default:
218             DEBUG("Invalid Timer Clock Divider selected\n");
219             tim->clockDivider = TIM_CKD_DIV1;
220             break;
221     }
222 }
```

Listing A.33: TIM1.h

```
1 #ifndef USER_TIM1_H_
2 #define USER_TIM1_H_
3
4 TIMER_t * TIM1_CH1_GetStruct(void);
5 void TIM1_CH1_Initialize(void);
6
7 #endif /* USER_TIM1_H_ */
```

Listing A.34: TIM1.c

```
#include "stm32f10x_tim.h"
#include "defines.h"
#include "NVIC.h"
#include "RCC.h"
#include "TIM.h"
#include "TIM1.h"

static TIMER_t timer1ch1;

//Get the TIM1 instance of the TIMER_t structure.
TIMER_t * TIM1_CH1_GetStruct(void)
{
   return &timer1ch1;
}

//Initialize TIM1 with the set parameters in the instance timer1ch1.
void TIM1_CH1_Initialize(void)
{
   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
   TIM_OCInitTypeDef TIM_OCInitStructure;

   //Configure TIM1 clocks.
   RCC_TIM1_Initialize();

   //Set timer parameters.
   TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);
   TIM_TimeBaseInitStructure.TIM_Prescaler = timer1ch1.prescaler;
   TIM_TimeBaseInitStructure.TIM_CounterMode = timer1ch1.counterMode;
   TIM_TimeBaseInitStructure.TIM_Period = timer1ch1.period;
   TIM_TimeBaseInitStructure.TIM_ClockDivision = timer1ch1.clockDivider;
   TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);

   //Set output compare parameters.
   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
   TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
   TIM_OCInitStructure.TIM_Pulse = (uint16_t) 0x7F;
   TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
   TIM_OC1Init(TIM1, &TIM_OCInitStructure);

   //Enable auto reload register.
   TIM_ARRPreloadConfig(TIM1, ENABLE);

   //Register TIM1 Capture Compare with NVIC.
   NVIC_TIM1_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_1);

   //Clear timer
   TIM_SetCounter(TIM1, 0);

   //Enable channel 1.
   TIM_CCxCmd(TIM1, TIM_Channel_1, TIM_CCx_Enable);

   //Enable capture/compare interrupt.
   TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);

   //Enable output signal to trigger ADC.
   TIM_CtrlPWMOutputs(TIM1, ENABLE);
}
```

Listing A.35: TIM2.h

```
1 #ifndef USER_TIM2_H_
2 #define USER_TIM2_H_
3
4 extern volatile uint32_t timer2Counter;
5
6 void TIM2_CH1_Initialize(void);
7 uint32_t TIM2_CH1_GetTick(void);
8
9 #endif /* USER_TIM2_H_ */
```

Listing A.36: TIM2.c

```
#include "stm32f10x_tim.h"
#include "defines.h"
#include "TIM.h"
#include "TIM2.h"
#include "NVIC.h"
#include "RCC.h"

volatile uint32_t timer2Counter = 0;

//Initilize TIM2 as a millisecond counter.
void TIM2_CH1_Initialize(void)
{
   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;

   //Configure TIM2 clocks.
   RCC_TIM2_Initialize();

   //Set timer parameters.
   //APB1 clock = 36 MHz.
   //1 ms resolution = 36 MHz / 36 000.
   TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);
   TIM_TimeBaseInitStructure.TIM_Prescaler = 36000;
   TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInitStructure.TIM_Period = 1;
   TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
   TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStructure);

   //Enable auto reload register.
   TIM_ARRPreloadConfig(TIM2, ENABLE);

   //Register TIM2 Capture Compare with NVIC.
   NVIC_TIM2_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);

   //Clear timer
   TIM_SetCounter(TIM2, 0);

   //Enable channel 1.
   TIM_CCxCmd(TIM2, TIM_Channel_1, TIM_CCx_Enable);

   //Enable capture/compare interrupt.
   TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);
}

//Get current tick from TIM2.
uint32_t TIM2_CH1_GetTick(void)
{
   return timer2Counter;
}
```

Listing A.37: TIM8.h

```
1 #ifndef USER_TIM8_H_
2 #define USER_TIM8_H_
3
4 TIMER_t * TIM8_CH1_GetStruct(void);
5 void TIM8_CH1_Initialize(void);
6
7 #endif /* USER_TIM8_H_ */
```

Listing A.38: TIM8.c

```
1  #include "stm32f10x_tim.h"
2  #include "defines.h"
3  #include "TIM.h"
4  #include "TIM8.h"
5  #include "NVIC.h"
6  #include "RCC.h"
7
8  static TIMER_t timer8ch1;
9
10 //Get the TIM8 instance of the TIMER_t structure.
11 TIMER_t * TIM8_CH1_GetStruct(void)
12 {
13    return &timer8ch1;
14 }
15
16 //Initialize TIM8 with the set parameters in the instance timer8ch1.
17 void TIM8_CH1_Initialize(void)
18 {
19    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
20    TIM_OCInitTypeDef TIM_OCInitStructure;
21
22    //Configure TIM8 clocks.
23    RCC_TIM8_Initialize();
24
25    //Set timer parameters.
26    TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);
27    TIM_TimeBaseInitStructure.TIM_Prescaler = timer8ch1.prescaler;
28    TIM_TimeBaseInitStructure.TIM_CounterMode = timer8ch1.counterMode;
29    TIM_TimeBaseInitStructure.TIM_Period = timer8ch1.period;
30    TIM_TimeBaseInitStructure.TIM_ClockDivision = timer8ch1.clockDivider;
31    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseInitStructure);
32
33    //Set output compare parameters.
34    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
35    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
36    TIM_OCInitStructure.TIM_Pulse = 0x7F;
37    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
38    TIM_OC1Init(TIM8, &TIM_OCInitStructure);
39
40    //Enable auto reload register.
41    TIM_ARRPreloadConfig(TIM8, ENABLE);
42
43    //Register TIM8 Capture Compare with NVIC.
44    NVIC_TIM8_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);
45
46    //Clear timer
47    TIM_SetCounter(TIM8, 0);
48
49    //Enable channel 1.
50    TIM_CCxCmd(TIM8, TIM_Channel_1, TIM_CCx_Enable);
51
52    //Enable capture/compare interrupt.
53    TIM_ITConfig(TIM8, TIM_IT_CC1, ENABLE);
54
55    //Enable output signal to trigger ADC.
56    TIM_CtrlPWMOutputs(TIM8, ENABLE);
57 }
```

Listing A.39: UART.h

```
#ifndef USER_UART_H_
#define USER_UART_H_

#include "Queues.h"

void USART1_Initialize(void);
void USART1_SendData(uint8_t data);
void USART1_SendDataArray(uint8_t * data, uint16_t length);
uint16_t USART1_ReceiveData(void);
void USART1_EnableReceiveData(void);
void USART1_DisableReceiveData(void);
Q8 * USART1_GetReceiveQueue(void);

#endif /* USER_UART_H_ */
```

Listing A.40: UART.c

```
1  #include "stm32f10x.h"
2  #include "defines.h"
3  #include "Queues.h"
4  #include "UART.h"
5  #include "stm32f10x_usart.h"
6  #include "GPIO.h"
7  #include "NVIC.h"
8  #include "RCC.h"
9
10 Q8 usartRxQueue;
11
12 //Initialize the USART1 for sending and receiving UART data at 115200 bps
       with
13 //no parity and one stop bit.
14 void USART1_Initialize(void)
15 {
16    USART_InitTypeDef USART_InitStructure;
17    USART_ClockInitTypeDef USART_ClockInitStructure;
18
19    //Configure USART1 clocks.
20    RCC_USART1_Initialize();
21
22    //Configure USART1 GPIO.
23    GPIO_USART1_Initialize();
24
25    //Register USART1 with NVIC.
26    NVIC_USART1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);
27
28       //USART
29    USART_Cmd(USART1, DISABLE);
30    USART_DeInit(USART1);
31    USART_InitStructure.USART_BaudRate = (uint32_t) USART1_BAUDRATE;
32    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
33    USART_InitStructure.USART_StopBits = USART_StopBits_1;
34    USART_InitStructure.USART_Parity = USART_Parity_No;
35    USART_InitStructure.USART_Mode = (USART_Mode_Tx | USART_Mode_Rx);
36    USART_InitStructure.USART_HardwareFlowControl =
          USART_HardwareFlowControl_None;
37    USART_Init(USART1, &USART_InitStructure);
38
39       //USART Clock
40    USART_ClockStructInit(&USART_ClockInitStructure);
41    USART_ClockInit(USART1, &USART_ClockInitStructure);
42
43    //Enable Peripheral
44    USART_Cmd(USART1, ENABLE);
45 }
46
47 //Send a byte of data.
48 void USART1_SendData(uint8_t data)
49 {
50    while( USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET );
51    USART_SendData(USART1, data);
52 }
53
54 //Send an array of data.
55 void USART1_SendDataArray(uint8_t * data, uint16_t length)
56 {
57    uint16_t i;
58
```

```
59      for(i = 0; i < length; i++)
60      {
61          while( USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET );
62          USART_SendData(USART1, data[i]);
63      }
64  }
65
66  //Wait for a byte of received data and return it.
67  uint16_t USART1_ReceiveData(void)
68  {
69      while( USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET );
70      return USART_ReceiveData(USART1);
71  }
72
73  //Enable asynchronously receiving data.
74  void USART1_EnableReceiveData(void)
75  {
76      //Enable receive data interrupt.
77      USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
78  }
79
80  //Disable asynchronously receiving data.
81  void USART1_DisableReceiveData(void)
82  {
83      //Disable receive interrupt.
84      USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
85  }
86
87  //Get a pointer to the UART receive queue.
88  Q8 * USART1_GetReceiveQueue(void)
89  {
90      return &usartRxQueue;
91  }
```

Listing A.41: ZCD.h

```
1 #ifndef USER_ZCD_H_
2 #define USER_ZCD_H_
3
4 void ZCD_Initialize(void);
5 void ZCD_Enable(void);
6 void ZCD_Disable(void);
7
8 #endif /* USER_ZCD_H_ */
```

Listing A.42: ZCD.c

```
#include "stm32f10x.h"
#include "defines.h"
#include "stm32f10x_exti.h"
#include "ZCD.h"
#include "RCC.h"
#include "GPIO.h"
#include "NVIC.h"
#include "EXTI.h"

//Initialize the ZCD.
void ZCD_Initialize(void)
{
   //Disable Zero Crossing by masking EXTI2 ISR.
   ZCD_Disable();

   //Configure ZDC clocks.
   RCC_ZCD_Initialize();

   //Configure ZDC GPIO.
   GPIO_ZCD_Initialize();

   //Configure zero crossing detector in NVIC.
   NVIC_ZCD_Initialize(NVIC_PRIORITY_2, NVIC_SUB_PRIORITY_0);

   //Configure zero crossing detector interrupt pin.
   EXTI_ZDC_Initialization();
}

//Enable the ZCD interrupt.
void ZCD_Enable(void)
{
   EXTI->IMR |= ZERO_CROSSING_DETECTOR_ENABLE_INTERRUPT_MASK;
}

//Mask/Disable the ZCD interrupt line.
void ZCD_Disable(void)
{
   EXTI->IMR &= ZERO_CROSSING_DETECTOR_DISABLE_INTERRUPT_MASK;
}
```

Listing A.43: Common.h

```
1 #ifndef USER_COMMON_H_
2 #define USER_COMMON_H_
3
4 uint32_t Common_ToXBits(uint32_t value, uint32_t maxValue, uint8_t bits);
5 uint32_t Common_FromXBits(uint32_t bitValue, uint8_t maxBits, uint32_t
      maxValue);
6 int32_t Common_FromXBitsSigned(int32_t bitValue, uint8_t maxBits, uint32_t
      maxValue);
7
8 #endif /* USER_COMMON_H_ */
```

Listing A.44: Common.c

```
1  #include <math.h>
2  #include <stdint.h>
3  #include "Common.h"
4  #include "defines.h"
5
6  //Convert value to a range of 0 to bits bit, where maxValue is the maximum
       value value can have. bits = x, where 2^x.
7  uint32_t Common_ToXBits(uint32_t value, uint32_t maxValue, uint8_t bits)
8  {
9     return ( (value * pow(2, bits)) / maxValue );
10 }
11
12 //Convert bitValue to a range from 0 to maxValue, where maxBits is the
       maximum bits bitValue can have. maxBits = x, where 2^x.
13 uint32_t Common_FromXBits(uint32_t bitValue, uint8_t maxBits, uint32_t
       maxValue)
14 {
15    return ( (bitValue * maxValue) / pow(2, maxBits) );
16 }
17
18 //Convert bitValue to a range from 0 to maxValue, where maxBits is the
       maximum bits bitValue can have. maxBits = x, where 2^x.
19 int32_t Common_FromXBitsSigned(int32_t bitValue, uint8_t maxBits, uint32_t
       maxValue)
20 {
21    return (int32_t) ( ( (double)bitValue * (double)maxValue) / pow((double)
          2, (double)maxBits) );
22 }
```

Listing A.45: DSP.h

```
#ifndef __DSP_H_
#define __DSP_H_

#include "arm_math.h"

//Global variables/structures
typedef struct
{
   int16_t buffer[MOVING_AVERAGE_BUFFER_SIZE];
   uint16_t oldestSample;
   int32_t sum;
} Queue_MA;

//Split data
void DSP_SplitToSignalAndReference(uint32_t * inputArray, uint16_t length,
    int16_t * signal, int16_t * reference);

//Minimum and maximum calculations.
int16_t DSP_FindMin_i16(int16_t * dataArray, uint16_t length);
int16_t DSP_FindMax_i16(int16_t * dataArray, uint16_t length);
uint16_t DSP_FindMax_ui16(uint16_t * dataArray, uint16_t length);
uint32_t DSP_FindMax_ui32(uint32_t * dataArray, uint16_t length);
uint16_t DSP_FindMin_ui16(uint16_t * dataArray, uint16_t length);
uint32_t DSP_FindMin_ui32(uint32_t * dataArray, uint16_t length);

//Saturation checks.
uint16_t DSP_CheckSaturationUpperAndMax(int16_t * dataArray, uint16_t length
    , int16_t maxValue, uint16_t nMaxValuesAllowed, int16_t * maxValueFound)
    ;
uint16_t DSP_CheckSaturationLowerAndMin(int16_t * dataArray, uint16_t length
    , int16_t minValue, uint16_t nMinValuesAllowed, int16_t * minValueFound)
    ;

//Averaging.
uint16_t DSP_AverageValueOfArray_ui16(uint16_t * dataArray, uint16_t length)
    ;
uint32_t DSP_AverageValueOfArray_ui32(uint32_t * dataArray, uint16_t length)
    ;
int16_t DSP_AverageValueOfArray_i16(int16_t * dataArray, uint16_t length);
int DSP_AverageValueOfArray_i32(int * dataArray, uint16_t length);
float DSP_AverageValueOfArray_i16_f(int16_t * dataArray, uint16_t length);
float DSP_ReferenceDutyCycle(int16_t * ref, uint16_t length);

//Sine wave operations.
void DSP_Rectify(int16_t * dataArray, uint16_t length);
float DSP_SineAverageToRms(float sineWaveAverage, uint16_t length);
double DSP_SinePeakToRms(double sineWavePeak);
double DSP_SinePeakToPeakToRms(double sineWavePeakToPeak);
float DSP_RmsToSinePeak(float rmsSine);
float DSP_RmsToSinePeakToPeak(float rmsSine);

//Rad <-> Degrees
double DSP_RadToDeg(double rad);
double DSP_DegToRad(double deg);

//Offset
void DSP_OffsetArray(int16_t * inputDataArray, uint16_t length, int16_t
    offset);

//Analog to digital array
```

```
53 void DSP_AnalogToDigitalArray(int16_t * inputArray, uint16_t length,
       uint16_t threshold);
54 uint16_t DSP_SquareDebounce(int16_t * ADCdataRef, uint16_t samples, uint8_t
       debounceLevel);
55 void DSP_ShiftSquareReference(int16_t * inputArray, uint16_t length, int16_t
        samplesToShift, uint16_t samplesPerPeriod);
56
57 //Higher level functions
58 void FIR_initializeNewInstanceAndStateArray_q31(arm_fir_instance_q31 *
       firInstance, q31_t * stateArray, q31_t * coefficients, uint16_t
       numberOfTaps, uint32_t blockSize);
59 void FIR_filterQ31(arm_fir_instance_q31 * firInstance, q31_t * inputArray,
       q31_t * outputArray, uint16_t samples, uint32_t blockSize);
60 void DSP_LockIn(arm_fir_instance_q31 * firInstance, int16_t * signal,
       int16_t * reference, uint16_t samples, q31_t * outputInPhase, q31_t *
       outputOutPhase);
61 void DSP_main(int16_t * ADCdataSignal, int16_t * ADCdataRef, q31_t * I,
       q31_t * Q);
62
63 //IQ demodulation functions.
64 void DSP_SamplesToIQ(int16_t * samples, int16_t * I, int16_t * Q, uint16_t
       periods);
65 void DSP_FourSamplesToIQ(int16_t * samples, int16_t * I, int16_t * Q);
66 double DSP_IQToAmplitude(int16_t I, int16_t Q);
67 double DSP_IQToPhase(int16_t I, int16_t Q);
68
69 //Moving average
70 void Q_MovingAverageFilterInitializeAverage(Queue_MA * queue, uint16_t
       average);
71 int16_t Q_MovingAverageFilter(Queue_MA * queue, int16_t newSample);
72
73 #endif /* __DSP_H_ */
```

Listing A.46: DSP.c

```
1  #include "defines.h"
2  #include "arm_math.h"
3  #include "DSP.h"
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <math.h>
8
9  //FIR filter
10 arm_fir_instance_q31 FIR_instance;
11 q31_t testStateArray[BLOCK_SIZE + NUMBER_OF_TAPS - 1];
12
13 //Temporary arrays for the lock-in mixer.
14 q31_t tempInPhase[NUMBER_OF_SAMPLES];
15 q31_t tempOutPhase[NUMBER_OF_SAMPLES];
16
17 //Fix point FIR filter coefficients.
18 q31_t FIR_coef[33] = {
19      122678030,   149432540,   231892874,   335975812,   461791746,
          608231181,
20      772800248,   951470521,  1139112950,  1329347733,  1515018811,
          1688838456,
21     1843296954,  1971539290,  2067705447,  2127323209,  2147483647,
          2127323209,
22     2067705447,  1971539290,  1843296954,  1688838456,  1515018811,
          1329347733,
23     1139112950,   951470521,   772800248,   608231181,   461791746,
          335975812,
24      231892874,   149432540,   122678030
25 };
26
27 //Shift an array of samples by samplesToShift.
28 void DSP_ShiftSquareReference(int16_t * inputArray, uint16_t length, int16_t
       samplesToShift, uint16_t samplesPerPeriod)
29 {
30    uint16_t i;
31    int16_t * tempArray;
32
33    if (samplesPerPeriod <= samplesToShift)
34    {
35       //Shift more than one period is not allowed. If samplesPerPeriod =
             samplesToShift, skip shift as it is already correctly "shifted".
36       return;
37    }
38
39    //Shift to the left?
40    if( samplesToShift > 0)
41    {
42       for(i = 0; i < length - samplesToShift; i++)
43       {
44          inputArray[i] = inputArray[i + samplesToShift];
45       }
46    }
47    //Shift to the right?
48    else if( samplesToShift < 0)
49    {
50       tempArray = malloc(sizeof(int16_t) * length);
51
52       memcpy(tempArray, inputArray, length * sizeof(int16_t));
53
```

```
54          for(i = 0; i < length - samplesToShift; i++)
55          {
56              inputArray[i + samplesToShift] = tempArray[i];
57          }
58
59          free(tempArray);
60      }
61  }
62
63  //Split the sampled signal and square reference signal data to two different
        arrays.
64  void DSP_SplitToSignalAndReference(uint32_t * inputArray, uint16_t length,
        int16_t * signal, int16_t * reference)
65  {
66      uint16_t i;
67
68      for(i = 0; i < length; i++)
69      {
70          signal[i] = ( ( inputArray[i] >> 16 ) & 0x0FFF);
71          reference[i] = (inputArray[i] & 0x0FFF);
72      }
73  }
74
75  //Convert from radians to degrees.
76  double DSP_RadToDeg(double rad)
77  {
78      double deg;
79
80      deg = rad * (180 / M_PI);
81
82      return deg;
83  }
84
85  //Convert from degrees to radians.
86  double DSP_DegToRad(double deg)
87  {
88      double rad;
89
90      rad = deg * (M_PI / 180);
91
92      return rad;
93  }
94
95  //Find maximum of an unsigned half-word.
96  uint16_t DSP_FindMax_ui16(uint16_t * dataArray, uint16_t length)
97  {
98      uint16_t max = 0;
99      uint16_t i;
100
101     for(i = 0; i < length; i++)
102     {
103         //Check if new value if greater than max
104         if(dataArray[i] > max)
105         {
106             max = dataArray[i];
107         }
108     }
109     return max;
110 }
111
112 //Find maximum of an unsigned word.
113 uint32_t DSP_FindMax_ui32(uint32_t * dataArray, uint16_t length)
```

```
114 {
115    uint32_t max = 0;
116    uint16_t i;
117 
118    for(i = 0; i < length; i++)
119    {
120       //Check if new value if greater than max
121       if(dataArray[i] > max)
122       {
123          max = dataArray[i];
124       }
125    }
126    return max;
127 }
128 
129 //Find minimum of an unsigned half-word.
130 uint16_t DSP_FindMin_ui16(uint16_t * dataArray, uint16_t length)
131 {
132    uint16_t min = 65535;
133    uint16_t i;
134 
135    for(i = 0; i < length; i++)
136    {
137       //Check if new value if less than min
138       if(dataArray[i] < min)
139       {
140          min = dataArray[i];
141       }
142    }
143    return min;
144 }
145 
146 //Find minimum of an unsigned word.
147 uint32_t DSP_FindMin_ui32(uint32_t * dataArray, uint16_t length)
148 {
149    uint32_t min = 65535;
150    uint16_t i;
151 
152    for(i = 0; i < length; i++)
153    {
154       //Check if new value if less than min
155       if(dataArray[i] < min)
156       {
157          min = dataArray[i];
158       }
159    }
160    return min;
161 }
162 
163 //Find minimum of a signed half-word.
164 int16_t DSP_FindMin_i16(int16_t * dataArray, uint16_t length)
165 {
166    int16_t min = 32767;
167    uint16_t i;
168 
169    for(i = 0; i < length; i++)
170    {
171       //Check if new value if less than min
172       if(dataArray[i] < min)
173       {
174          min = dataArray[i];
175       }
```

```
176     }
177     return min;
178 }
179 
180 //Find maximum of a signed half-word.
181 int16_t DSP_FindMax_i16(int16_t * dataArray, uint16_t length)
182 {
183     int16_t max = -32768;
184     uint16_t i;
185 
186     for(i = 0; i < length; i++)
187     {
188         //Check if new value if greater than max
189         if(dataArray[i] > max)
190         {
191             max = dataArray[i];
192         }
193     }
194     return max;
195 }
196 
197 //Check for maximum saturation in an array by determining a threshold and
        how many values at or above this threshold is allowed.
198 uint16_t DSP_CheckSaturationUpperAndMax(int16_t * dataArray, uint16_t length
        , int16_t maxValue, uint16_t nMaxValuesAllowed, int16_t * maxValueFound)
199 {
200     int16_t max = -32768;
201     uint16_t maxValueOccurenceCounter = 0, i;
202 
203     //Loop through array
204     for(i = 0; i < length; i++)
205     {
206         //Check if value is greater or equal to threshold value maxValue
207         if(dataArray[i] >= maxValue)
208         {
209             maxValueOccurenceCounter++;
210             if(maxValueOccurenceCounter > nMaxValuesAllowed)
211             {
212                 *maxValueFound = (max > dataArray[i]) ? max : dataArray[i];
213                 return 0xFFFF;
214             }
215 
216             max = dataArray[i];
217         }
218           else if(max < dataArray[i])
219           {
220               max = dataArray[i];
221           }
222     }
223 
224     //Write maximum value
225     *maxValueFound = max;
226 
227     //Return number of values equal or greater than maxValue
228     return maxValueOccurenceCounter;
229 }
230 
231 //Check for minimum saturation in an array by determining a threshold and
        how many values at or below this threshold is allowed.
232 uint16_t DSP_CheckSaturationLowerAndMin(int16_t * dataArray, uint16_t length
        , int16_t minValue, uint16_t nMinValuesAllowed, int16_t * minValueFound)
233 {
```

```
234     int16_t min = 32767;
235     uint16_t minValueOccurenceCounter = 0, i;
236 
237     //Loop through array
238     for(i = 0; i < length; i++)
239     {
240        //Check if value is less or equal to threshold value minValue
241        if(dataArray[i] <= minValue)
242        {
243           minValueOccurenceCounter++;
244           if(minValueOccurenceCounter > nMinValuesAllowed)
245           {
246              //Allowed number of min values reached, skip rest of computation
                      and return error.
247              *minValueFound = (min < dataArray[i]) ? min : dataArray[i];
248              return 0xFFFF;
249           }
250           min = dataArray[i];
251        }
252          else if (min > dataArray[i])
253          {
254              min = dataArray[i];
255          }
256     }
257 
258     //Write minimum value
259     *minValueFound = min;
260 
261     //Return number of values equal or less than minValue
262     return minValueOccurenceCounter;
263 }
264 
265 //Find average of a unsigned half-word array, rounds down.
266 uint16_t DSP_AverageValueOfArray_ui16(uint16_t * dataArray, uint16_t length)
267 {
268     uint16_t i, average;
269 
270     //Sum can overflow if all inputs are 65535 and length is 65535 (unlikly),
          but not else.
271     uint32_t sum = 0;
272 
273     //Calculate average.
274     for(i = 0; i < length; i++)
275     {
276        sum += dataArray[i];
277     }
278 
279     //Find average.
280     average = sum/length;
281 
282     return average;
283 }
284 
285 //Find average of a unsigned word array, rounds down.
286 uint32_t DSP_AverageValueOfArray_ui32(uint32_t * dataArray, uint16_t length)
287 {
288     uint16_t i;
289     uint32_t average;
290     uint64_t sum = 0;
291 
292     //Calculate average.
293     for(i = 0; i < length; i++)
```

```
294     {
295         sum += dataArray[i];
296     }
297 
298     //Calculate average.
299     average = (uint32_t) (sum/length);
300 
301     return average;
302 }
303 
304 //Find average of a signed half-word array, rounds down.
305 int16_t DSP_AverageValueOfArray_i16(int16_t * dataArray, uint16_t length)
306 {
307     uint16_t i;
308     int16_t average;
309     int32_t sum = 0;
310 
311     //Calculate average.
312     for(i = 0; i < length; i++)
313     {
314         sum += dataArray[i];
315     }
316 
317     //Calculate average.
318     average = sum/length;
319 
320     return average;
321 }
322 
323 //Find average of a signed word array, rounds down.
324 int DSP_AverageValueOfArray_i32(int * dataArray, uint16_t length)
325 {
326     uint16_t i;
327     int average;
328     int64_t sum = 0;
329 
330     //Calculate average.
331     for(i = 0; i < length; i++)
332     {
333         sum += dataArray[i];
334     }
335 
336     //Calculate average.
337     average = (int) (sum/length);
338 
339     return average;
340 }
341 
342 //Find average of a float array.
343 float DSP_AverageValueOfArray_i16_f(int16_t * dataArray, uint16_t length)
344 {
345     uint16_t i;
346     float average;
347     int32_t sum = 0;
348 
349     //Calculate average.
350     for(i = 0; i < length; i++)
351     {
352         sum += dataArray[i];
353     }
354 
355     //Calculate average.
```

```
356     average = ( (float) sum / (float) length );
357
358     return average;
359 }
360
361 //Calculate the duty cycle of a 1-bit signal.
362 float DSP_ReferenceDutyCycle(int16_t * ref, uint16_t length)
363 {
364     uint16_t i;
365     float dutyCycle;
366     int32_t counter = 0;
367
368     //Calculate Duty Cycle
369     for(i = 0; i < length; i++)
370     {
371         if( ref[i] == 1)
372         {
373             counter++;
374         }
375     }
376
377     dutyCycle = ( (float) counter / (float) length );
378
379     return dutyCycle;
380 }
381
382 //Full wave rectifier.
383 void DSP_Rectify(int16_t * dataArray, uint16_t length)
384 {
385     uint16_t i;
386
387     for (i = 0; i < length; i++)
388     {
389         if(dataArray[i] < 0)
390         {
391             dataArray[i] *= -1;
392         }
393     }
394 }
395
396 //Convert from sinusoidal average to RMS.
397 float DSP_SineAverageToRms(float sineWaveAverage, uint16_t length)
398 {
399     return (V_AVG_TO_RMS * sineWaveAverage);
400 }
401
402 //Convert from sinusoidal peak to RMS.
403 double DSP_SinePeakToRms(double sineWavePeak)
404 {
405     //Vrms = ( 1 / sqrt(2) ) * Vpeak
406     return (M_SQRT1_2 * sineWavePeak);
407 }
408
409 //Convert from sinusoidal peak-to-peak to RMS.
410 double DSP_SinePeakToPeakToRms(double sineWavePeakToPeak)
411 {
412     //Vrms = ( 1 / sqrt(2) ) * (Vpeak-peak / 2)
413     return ((M_SQRT1_2 * sineWavePeakToPeak) / 2);
414 }
415
416 //Convert from RMS to sinusoidal peak value.
417 float DSP_RmsToSinePeak(float rmsSine)
```

```
418 {
419    //Vpeak = Vrms * sqrt(2)
420    return rmsSine * M_SQRT2;
421 }
422
423 //Convert from RMS to sinusoidal peak-to-peak.
424 float DSP_RmsToSinePeakToPeak(float rmsSine)
425 {
426    //Vpeak-peak = Vrms * sqrt(2) * 2
427    return rmsSine * M_SQRT2 * 2;
428 }
429
430 //Offset an array with offset.
431 void DSP_OffsetArray(int16_t * inputDataArray, uint16_t length, int16_t
       offset)
432 {
433    uint16_t i;
434
435    for(i = 0; i < length; i++)
436    {
437       inputDataArray[i] -= offset;
438    }
439 }
440
441 //Converts a signal to a 1-bit signal with the threshold threshold.
442 void DSP_AnalogToDigitalArray(int16_t * inputArray, uint16_t length,
       uint16_t threshold)
443 {
444    uint16_t i;
445
446    for(i = 0; i < length; i++)
447    {
448       //Give input array boolean values. This depends if the current value
           is above or below given threshold
449         if( ((i) % (SAMPLING_FACTOR/2)) == 0 )
450         {
451             if (i == 0)
452             {
453                 continue;
454             }
455             inputArray[i] = -1;
456         }
457         else if(inputArray[i] >= threshold)
458       {
459          inputArray[i] = 1;
460       }
461       else
462       {
463          inputArray[i] = -1;
464       }
465    }
466 }
467
468 //Digital debouncing filter.
469 uint16_t DSP_SquareDebounce(int16_t * ADCdataRef, uint16_t samples, uint8_t
       debounceLevel)
470 {
471    uint8_t j;
472    uint16_t i, debounceCounter = 0;
473
474    //Loop through samples
475    for(i = 1; i < samples; i++)
```

```
476     {
477        //Check for change in level
478        if( ADCdataRef[i] != ADCdataRef[i - 1] )
479        {
480           //Apply debounce filter
481           for(j = 0; j < debounceLevel; j++)
482           {
483              if( ADCdataRef[i + j + 1] != ADCdataRef[i] )
484              {
485                 ADCdataRef[i] = ADCdataRef[i - 1];
486                 debounceCounter++;
487              }
488           }
489        }
490     }
491     return debounceCounter;
492  }
493
494  //Initialize a FIR filter instance.
495  void FIR_initializeNewInstanceAndStateArray_q31(arm_fir_instance_q31 *
         firInstance, q31_t * stateArray, q31_t * coefficients, uint16_t
         numberOfTaps, uint32_t blockSize)
496  {
497     arm_fir_init_q31(firInstance, numberOfTaps, coefficients, stateArray,
           blockSize);
498  }
499
500  //FIR filter Q1.31 fixed-point number values.
501  void FIR_filterQ31(arm_fir_instance_q31 * firInstance, q31_t * inputArray,
         q31_t * outputArray, uint16_t samples, uint32_t blockSize)
502  {
503     //Variables
504     uint16_t i;
505
506     for(i = 0; i < (samples/blockSize); i++)
507     {
508        arm_fir_q31(firInstance, &inputArray[0] + (i * blockSize), &
              outputArray[0] + (i * blockSize), blockSize);
509     }
510  }
511
512  //Lock-in amplifier a signal and reference with a FIR low pass filter.
513  void DSP_LockIn(arm_fir_instance_q31 * firInstance, int16_t * signal,
         int16_t * reference, uint16_t samples, q31_t * outputInPhase, q31_t *
         outputOutPhase)
514  {
515     uint16_t i;
516
517     //Mix signal and ref. 0 and 90 degree ref.
518     for(i = 0; i < samples - (SAMPLING_FACTOR / 4); i++)
519     {
520        tempInPhase[i] = (q31_t) (signal[i] * reference[i]);
521        tempOutPhase[i] = (q31_t) (signal[i] * reference[i + (SAMPLING_FACTOR
              / 4) ]);
522     }
523
524     //FIR filter 2f component in product, DC should be remaining
525     FIR_filterQ31(firInstance, &tempInPhase[0], &outputInPhase[0], samples -
           (SAMPLING_FACTOR / 4), BLOCK_SIZE);
526     FIR_filterQ31(firInstance, &tempOutPhase[0], &outputOutPhase[0], samples
           - (SAMPLING_FACTOR / 4), BLOCK_SIZE);
527  }
```

```
528
529 //Prepare sampled data for lock-in, do lock-in with FIR filter and return
        the IQ demodulated values.
530 void DSP_main(int16_t * ADCdataSignal, int16_t * ADCdataRef, q31_t * I,
        q31_t * Q)
531 {
532    uint16_t debounceCounter = 0;
533    int16_t IMax, IMin, average;
534    float averageFloat;
535
536    //Check for upper and lower saturation on signal
537    if(DSP_CheckSaturationUpperAndMax(&ADCdataSignal[0], NUMBER_OF_SAMPLES,
           4095, NUMBER_OF_PERIODS, &IMax) == 0xFF)
538    {
539       DEBUG("ERROR: DSP_CheckSaturationUpper: FAILED\n\n");
540    }
541    else
542    {
543       DEBUG("DSP_CheckSaturationUpper: PASSED\n\n");
544    }
545
546    DEBUG("IMax: %i\n\n", IMax);
547
548    if( DSP_CheckSaturationLowerAndMin(&ADCdataSignal[0], NUMBER_OF_SAMPLES,
           0, NUMBER_OF_PERIODS, &IMin) == 0xFF)
549    {
550       DEBUG("ERROR: DSP_CheckSaturationLower: FAILED\n\n");
551    }
552    else
553    {
554       DEBUG("DSP_CheckSaturationLower: PASSED\n\n");
555    }
556
557    DEBUG("IMin: %i\n\n", IMin);
558
559    //Convert reference array to boolean values
560    DSP_AnalogToDigitalArray(&ADCdataRef[0], NUMBER_OF_SAMPLES,
           REF_DATA_ANALOG_TO_DIGITAL_THRESHOLD);
561    DEBUG("DSP_AnalogToDigitalArray: DONE\n\n");
562
563    //Create a phase shift on sqaure reference by shifting samples
564    DSP_ShiftSquareReference(&ADCdataRef[0], NUMBER_OF_SAMPLES,
           SHIFT_REFERENCE_BY_SAMPLES, SAMPLING_FACTOR);
565
566    //Debounce the square reference signal.
567    debounceCounter = DSP_SquareDebounce(&ADCdataRef[0], NUMBER_OF_SAMPLES,
           1);
568    DEBUG("Debounced samples: %u\n", debounceCounter);
569
570    //Get duty cycle of the square reference signal.
571    averageFloat = DSP_ReferenceDutyCycle(&ADCdataRef[0], NUMBER_OF_SAMPLES);
572
573    //Check if reference duty cycle is within some limits
574    if(REF_DUTY_CYCLE_LOWER_THRESHOLD <= averageFloat && averageFloat <=
           REF_DUTY_CYCLE_UPPER_THRESHOLD)
575    {
576       DEBUG("Reference duty cycle within limits %1.2f <-> %1.2f: %1.3f\n\n",
              REF_DUTY_CYCLE_LOWER_THRESHOLD, REF_DUTY_CYCLE_UPPER_THRESHOLD,
              averageFloat);
577    }
578    else
579    {
```

```
580         DEBUG("ERROR: Reference duty cycle outside limits %1.2f <-> %1.2f:
                %1.3f\n\n", REF_DUTY_CYCLE_LOWER_THRESHOLD,
                REF_DUTY_CYCLE_UPPER_THRESHOLD, averageFloat);
581     }
582
583     //Get signal array average
584     average = DSP_AverageValueOfArray_i16(&ADCdataSignal[0],
            NUMBER_OF_SAMPLES);
585
586     DEBUG("Offset / DC: %i\n\n", average);
587
588     //Remove offset (minimum) and get average to be y = 0
589     DSP_OffsetArray(&ADCdataSignal[0], NUMBER_OF_SAMPLES, average);
590
591     //Line below only for debugging
592     average = DSP_AverageValueOfArray_i16(&ADCdataSignal[0],
            NUMBER_OF_SAMPLES);
593
594     DEBUG("Signal offset after Offset / DC corrections: %i\n\n", average);
595
596     FIR_initializeNewInstanceAndStateArray_q31(&FIR_instance, &testStateArray
            [0], &FIR_coef[0], NUMBER_OF_TAPS, BLOCK_SIZE);
597
598     //Get I and Q
599     DSP_LockIn(&FIR_instance, &ADCdataSignal[0], &ADCdataRef[0],
            NUMBER_OF_SAMPLES, &I[0], &Q[0]);
600
601     return;
602 }
603
604 //Convert an array of samples from 4fs methods to I and Q values.
605 void DSP_SamplesToIQ(int16_t * samples, int16_t * I, int16_t * Q, uint16_t
        periods)
606 {
607     uint16_t i;
608
609     for(i = 0; i < periods; i++)
610     {
611         DSP_FourSamplesToIQ(&samples[(i << 2)], &I[i], &Q[i]);
612     }
613 }
614
615 //Convert 4 samples in a period using 4fs method calculate the I and Q value
        .
616 void DSP_FourSamplesToIQ(int16_t * samples, int16_t * I, int16_t * Q)
617 {
618     *I = samples[0] - samples[2];
619     *Q = samples[3] - samples[1];
620 }
621
622 //Calculate the amplitude from the I and Q value.
623 double DSP_IQToAmplitude(int16_t I, int16_t Q)
624 {
625     double amplitude;
626
627     amplitude = sqrt( (double)I*(double)I + (double)Q*(double)Q );
628
629     return amplitude;
630 }
631
632 //Calculate the phase from the I and Q value.
633 //Returns the phase with -90 degrees offset.
```

```
634 double DSP_IQToPhase(int16_t I, int16_t Q)
635 {
636    double phase;
637
638    phase = DSP_RadToDeg(atan2((double) Q, (double) I));
639
640    return phase;
641 }
642
643 //Used to set initial average value in a moving average filter.
644 void Q_MovingAverageFilterInitializeAverage(Queue_MA * queue, uint16_t
       average)
645 {
646    uint16_t i;
647
648    for(i = 0; i < MOVING_AVERAGE_BUFFER_SIZE; i++)
649    {
650       queue->buffer[i] = average;
651    }
652
653    return;
654 }
655
656 //Moving average filter.
657 int16_t Q_MovingAverageFilter(Queue_MA * queue, int16_t newSample)
658 {
659    //Subtract oldest sample from sum.
660    queue->sum -= queue->buffer[queue->oldestSample];
661
662    //Add new sample to sum.
663    queue->sum += newSample;
664
665    //Replace the oldest sample with new sample in buffer.
666    queue->buffer[queue->oldestSample] = newSample;
667
668    //Increment oldest sample index.
669    queue->oldestSample++;
670
671    //Mask oldest sample index so it wraps to start of buffer when end of
           buffer has been reached.
672    //MOVING_AVERAGE_BUFFER_SIZE = 2^n where n is an integer.
           MOVING_AVERAGE_BUFFER_SIZE will then for n = 8 be 256 = 0x100.
673    //If oldestSample is 0x100 and MOVING_AVERAGE_BUFFER_SIZE - 1 is 0x0FF,
           then 0x100 & 0x0FF = 0x000.
674    queue->oldestSample &= MOVING_AVERAGE_BUFFER_SIZE_MASK;
675
676    //Return the average of the last MOVING_AVERAGE_BUFFER_SIZE samples.
677    return (int16_t) (queue->sum / MOVING_AVERAGE_BUFFER_SIZE);
678 }
```

Listing A.47: Initialize.h

```
1 #ifndef USER_INITIALIZE_H_
2 #define USER_INITIALIZE_H_
3
4 void Initialize(void);
5
6 #endif /* USER_INITIALIZE_H_ */
```

Listing A.48: Initialize.c

```
1  #include "Initialize.h"
2  #include "defines.h"
3  #include "LED.h"
4  #include "IWDG.h"
5  #include "NVIC.h"
6  #include "TIM2.h"
7  #include "SYSTICK.h"
8  #include "UART.h"
9
10 //A collection of initializations to do at system start.
11 //Also checks whether the system was powered on normally or if the
12 //independent watchdog triggered a reset.
13 void Initialize(void)
14 {
15    //Set how many bits to be used for pre-emp and subpriority levels.
16    //Set two bits of each (4 levels each).
17    NVIC_Initialize();
18
19    //Initialize UART communication.
20    USART1_Initialize();
21
22    //Initialize System Timer.
23    SYSTICK_Initialize();
24
25    //Initialize Light Emitting Diodes.
26    LED_Initialize();
27
28    //Initialize Timer 2.
29    //TIM2_CH1_Initialize();
30
31    if(IWDG_hasResetHappened())
32    {
33       DEBUG("\nRESET");
34    }
35    else
36    {
37       DEBUG("\nNO RESET");
38    }
39
40    //Initialize Watchdog.
41    //WatchDog_Init();
42
43    //Enable watchdog.
44    //WatchDog_Start();
45 }
```

Listing A.49: Measurement.h

```
#ifndef USER_MEASUREMENT_H_
#define USER_MEASUREMENT_H_

//Calculations.
void Measure_Calculate_SamplingParameters(uint32_t signalFrequency, uint32_t samplingFrequency, bool_t calculate4Fs);
void Measure_Calculate_IQPhaseAmplitude(bool_t calculate4Fs);

//Calibration.
void Measure_Calibrate_CalculatePhaseErrorSlope(uint32_t frequency1, uint32_t frequency2);
double Measure_Calibrate_CorrectPhase(uint32_t measurementFrequency);
void Measure_Calibrate_SystemVoltage(void);
uint16_t Measure_GetSystemVoltage(void);

//Excitation.
void Measure_Excitation_UpdateParameters(uint32_t signalFrequency);
int16_t Measure_Excitation_GetDcOffset(void);
void Measure_SetExcitationPeakToPeakMillivolts(uint16_t exPtoPmV);

//4fs measurement.
void Measure_4fs_CSFM(uint32_t signalFrequency, uint8_t periodsToSample);
void Measure_4fs_Spectroscopy(unsigned int * frequencies, size_t numberOfFrequencies, uint8_t periodsToSample);
int16_t * Measure_4fs_GetDataArrayPointer(void);
int16_t * Measure_4fs_SetTempDataArrayPointer(int16_t * samplesPointer);
int16_t * Measure_4fs_GetTempDataArrayPointer();
void Measure_4fs_SetNumberOfSamples(uint16_t samplesToSet);
uint16_t Measure_4fs_GetNumberOfSamples(void);

//Normal measurement.
void Measure_Normal_Initialize(void);
void Measure_Normal_Start(uint32_t signalFrequency, uint32_t samplingFrequency, uint8_t periodsToSample);

//Immittance.
double Immittance_Calculate_IToCurrentMilliAmpere(int16_t I);
double Immittance_Calculate_QToCurrentMilliAmpere(int16_t Q);
double Immittance_Calculate_IQToCurrentMilliAmpere(int16_t IorQ);
double Immittance_Calculate_CurrentPeakToPeakMilliAmpere(void);
double Immittance_GetCurrentPeakToPeakMilliAmpere(void);
double Immittance_GetCurrentIMilliAmpere(void);
double Immittance_GetCurrentQMilliAmpere(void);
double Immittance_GetCurrentIMicroAmpere(void);
double Immittance_GetCurrentQMicroAmpere(void);
double Immittance_GetCurrentINanoAmpere(void);
double Immittance_GetCurrentQNanoAmpere(void);
void Immittance_Calculate_ImmittanceParameters(void);
double Immittance_GetPhase(void);
double Immittance_GetImpedanceModulus(void);
double Immittance_GetAdmittanceModulusInMicroSiemens(void);
double Immittance_GetResistance(void);
double Immittance_GetReactance(void);
double Immittance_GetConductanceInMicroSiemens(void);
double Immittance_GetSusceptanceInMicroSiemens(void);
void Immittance_Start(void);

#endif /* USER_MEASUREMENT_H_ */
```

Listing A.50: Measurement.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "stm32f10x.h"
#include "defines.h"
#include "Common.h"
#include "Measurement.h"
#include "Timer.h"
#include "TIM.h"
#include "TIM1.h"
#include "TIM8.h"
#include "ADC.h"
#include "ADC1.h"
#include "ADC2.h"
#include "ADC3.h"
#include "DDS.h"
#include "DMA.h"
#include "DSP.h"
#include "NVIC.h"
#include "ZCD.h"

//Excitation
static float exRms;
static int16_t exDcOffset;
static float exRectifiedAvg;
static uint16_t exRectifiedAvgMillivolts;
static uint16_t exPeakToPeakMillivolts;
static float exAmplitudeMillivolts;
static uint16_t exRmsMillivolts;

//IQ
int16_t I[MEASURE_MAX_SAMPLING_PERIODS];
static int16_t IAvg;
static int16_t IMin;
static int16_t IMax;

int16_t Q[MEASURE_MAX_SAMPLING_PERIODS];
static int16_t QAvg;
static int16_t QMin;
static int16_t QMax;

static double IQPeakToPeak;
static uint16_t IQPeakToPeakMilliVolts;
static double IQAmplitude;
static uint16_t IQAmplitudeMilliVolts;
static double IQPhase;
static double IQRmsMilliVolts;

//Current
static double iPeakToPeakMilliAmpere;
static double iI;
static double iQ;

//Immittance
static double Z;
static double Y;
static double R;
static double X;
static double G;
static double B;
```

```
61  static double phase;
62
63  //Data buffer
64  int16_t samples[MEASURE_SAMPLE_ARRAY_SIZE];
65  static int16_t * samplesPtr = &samples[0];
66  static int16_t * samplesTempPtr = &samples[0];
67  static uint16_t numberOfSamples;
68
69  //Sampling parameters
70  static uint32_t timerRatio;
71  static uint16_t timerPrescaler;
72  static uint16_t timerPeriod;
73  static float etsFs;
74  static uint32_t etsPeriods;
75  static uint32_t fs;
76  static uint32_t periods;
77  static uint32_t ratio;
78
79  //Calibration values
80  static uint16_t Vcc_MilliVolt = 3348;
81  static double calibrationPhaseSlope;
82  static uint32_t calibrationZeroPhaseFrequency = 0;
83
84  //Start a 4fs method measurement with frequency signalFrequency and
        periodsToSample periods.
85  static void Measure_4fs_Start(uint32_t signalFrequency, uint8_t
        periodsToSample)
86  {
87     //Set global periods parameter for later use.
88     periods = periodsToSample;
89
90     //Update DDS frequency if needed.
91     if(signalFrequency != DDS_GetFrequency())
92     {
93        DDS_Config(signalFrequency, 0, FALSE);
94     }
95
96     //Set DMA parameters
97     DMA_SetTransferDone(DMA_Get4fsStruct(), FALSE);
98
99     //Set sampling buffer.
100    DMA_SetSamplingBuffer(DMA_Get4fsStruct(), &samples[0]);
101
102    //Set number of samples.
103    DMA_SetNumberOfSamples(DMA_Get4fsStruct(), (4 * ((uint16_t)
           periodsToSample)) + 4);
104
105    //First four samples are tossed due to ADC response start time (usually
           only first sample is invalid).
106    Measure_4fs_SetTempDataArrayPointer(&samples[4]);
107
108    //Set number of valid samples.
109    Measure_4fs_SetNumberOfSamples(4 * periodsToSample);
110
111    //Initialize DMA.
112    DMA_ADC1_4fs_Initialize();
113
114    //Calculate timer parameters.
115    Measure_Calculate_SamplingParameters(DDS_GetFrequency(), (
           DDS_GetFrequency() * 4), TRUE);
116
117    //Configure timer.
```

```
118     TIM_Disable(TIM_1);
119     TIM_SetPrescaler(TIM1_CH1_GetStruct(), timerPrescaler);
120     TIM_SetPeriod(TIM1_CH1_GetStruct(), timerPeriod);
121     TIM_SetClockDivider(TIM1_CH1_GetStruct(), TIM_CLK_DIV_1);
122     TIM_SetCounterMode(TIM1_CH1_GetStruct(), TIM_CNT_MODE_UP);
123     TIM1_CH1_Initialize();
124 
125     //Enable DMA and ADC.
126     DMA_ADC1_Enable();
127     ADC_Enable(ADC_1);
128 
129     //Start conversion and enable timer on first falling edge.
130     ZCD_Enable();
131 
132     //Wait for 4*periods samples have been acquired.
133     while( DMA_IsTransferDone(DMA_Get4fsStruct()) == FALSE );
134 
135     //Disable ADC and DMA.
136     DMA_ADC1_Disable();
137 }
138 
139 //Start a 4fs method single frequency measurement, calculate current and
        immittance values.
140 void Measure_4fs_CSFM(uint32_t signalFrequency, uint8_t periodsToSample)
141 {
142     //Start 4fs measurement.
143     Measure_4fs_Start(signalFrequency, periodsToSample);
144 
145     //Calculate I, Q and phase parameters.
146     Measure_Calculate_IQPhaseAmplitude(TRUE);
147 
148     //Calculate currents and immittance parameters.
149     Immittance_Start();
150 }
151 
152 //Start a 4fs method quasi-parallel spectroscopy.
153 void Measure_4fs_Spectroscopy(unsigned int * frequencies, size_t
        numberOfFrequencies, uint8_t periodsToSample)
154 {
155     uint16_t i;
156     uint8_t j;
157     static uint16_t counter = 0;
158     double rTemp;
159     double xTemp;
160 
161     for(i = counter; i < (counter + 4); i++)
162     {
163 
164         rTemp = 0;
165         xTemp = 0;
166 
167         for(j = 0; j < 3; j++)
168         {
169             Measure_4fs_CSFM(frequencies[i], periodsToSample);
170 
171             rTemp += Immittance_GetResistance();
172             xTemp += Immittance_GetReactance();
173         }
174 
175         rTemp /= 3;
176         xTemp /= 3;
177 
```

```
178         //Send data to receiver.
179         SEND_MSG_SPECTROSCOPY_RESISTANCE_REACTANCE(frequencies[i], rTemp,
                xTemp);
180
181         //If sweep over all frequencies are done, start over.
182         if( i >= (numberOfFrequencies - 1) )
183         {
184            counter = 0;
185            return;
186         }
187      }
188
189      //Increment counter for next round.
190      counter +=4;
191   }
192
193   //Initialize normal lock-in mode.
194   void Measure_Normal_Initialize(void)
195   {
196      //Initialize DMA for ADC1 normal lock-in mode
197      DMA_ADC1_Normal_Initialize();
198
199      //Configure DMA 1 channel 1 in NVIC.
200      NVIC_DMA1_CH1_Initialize(NVIC_PRIORITY_3, NVIC_SUB_PRIORITY_3);
201
202      //Initialize ADC1 and ADC2 in normal lock-in mode.
203      ADC1_Normal_Initialize();
204      ADC2_Normal_Initialize();
205
206      //Enable ADC1 and ADC2 to start conversion on external trigger.
207      ADC_ExternalTriggerEnable(ADC_1);
208      ADC_ExternalTriggerEnable(ADC_2);
209
210      //Enable ADC2 and ADC1.
211      ADC_Enable(ADC_2);
212      ADC_Enable(ADC_1);
213
214      //Calibrate ADC1 and ADC2.
215      ADC_Calibrate(ADC_1);
216      ADC_Calibrate(ADC_2);
217
218      //Disable ADC1 and 2.
219      ADC_Disable(ADC_1);
220      ADC_Disable(ADC_2);
221   }
222
223   //Start a normal lock-in mode measurement.
224   void Measure_Normal_Start(uint32_t signalFrequency, uint32_t
        samplingFrequency, uint8_t periodsToSample)
225   {
226      uint32_t ratio;
227
228      periods = periodsToSample;
229
230      //Calculate sampling frequency to signal frequency ratio.
231      ratio = (uint32_t) (samplingFrequency / signalFrequency);
232
233      if(signalFrequency != DDS_GetFrequency())
234      {
235         DDS_Config(signalFrequency, 0, FALSE);
236      }
237
```

```
238     //Set DMA parameters and initialize.
239     DMA_SetTransferDone(DMA_GetNormalStruct(), FALSE);
240     DMA_SetSamplingBuffer(DMA_GetNormalStruct(), &samples[0]);
241     DMA_SetNumberOfSamples(DMA_GetNormalStruct(), ratio * periodsToSample);
242 
243     //Set up DMA.
244     DMA_ADC1_Normal_Initialize();
245 
246     //Calculate timer parameters.
247     Measure_Calculate_SamplingParameters(DDS_GetFrequency(), DDS_GetFrequency
            (), FALSE);
248 
249     //Configure timer.
250     TIM_Disable(TIM_1);
251     TIM_SetPrescaler(TIM1_CH1_GetStruct(), timerPrescaler);
252     TIM_SetPeriod(TIM1_CH1_GetStruct(), timerPeriod);
253     TIM_SetClockDivider(TIM1_CH1_GetStruct(), TIM_CLK_DIV_1);
254     TIM_SetCounterMode(TIM1_CH1_GetStruct(), TIM_CNT_MODE_UP);
255     TIM1_CH1_Initialize();
256 
257     //Enable DMA and ADC.
258     DMA_ADC1_Enable();
259     ADC_Enable(ADC_2);
260     ADC_Enable(ADC_1);
261 
262     TIM_Enable(TIM_1);
263 
264     //Wait for ratio * periods samples have been acquired.
265     while( DMA_IsTransferDone(DMA_Get4fsStruct()) == FALSE );
266 
267     //Disable ADC and DMA.
268     ADC_Disable(ADC_2);
269     ADC_Disable(ADC_1);
270     DMA_ADC1_Disable();
271 }
272 
273 void Measure_Calculate_SamplingParameters(uint32_t signalFrequency, uint32_t
        samplingFrequency, bool_t calculate4Fs)
274 {
275     fs = samplingFrequency;
276     etsFs = (float)samplingFrequency;
277 
278     if(calculate4Fs)
279     {
280         //4fs
281         if (samplingFrequency < ADC_MAX_SAMPLING_SPEED)
282         {
283             //Can sample all within one period.
284 
285             //Calculate timer parameters.
286             timerRatio = (SYSTEM_FREQUENCY / fs);
287 
288             //Scale prescaler as little as possible to maintain largest
                    possible resolution.
289             timerPrescaler = (uint16_t) (timerRatio / 65535);
290 
291             //Set timer period based on new prescaling.
292             timerPeriod = (timerRatio - (timerPrescaler * 65535));
293         }
294         else
295         {
296             //ETS sampling frequency.
```

```
297         }
298     }
299 }
300 
301 //Two point phase calibration.
302 void Measure_Calibrate_CalculatePhaseErrorSlope(uint32_t frequency1,
        uint32_t frequency2)
303 {
304     int i;
305     double phase1;
306     double phase2;
307     double deltaPhase;
308     uint32_t fOffset;
309 
310     //Dummy sample.
311     Measure_4fs_CSFM(frequency2, 10);
312 
313     phase2 = 0;
314 
315     //Get phase at frequency 2.
316     for(i = 0; i < 100; i++)
317     {
318         Measure_4fs_CSFM(frequency2, 10);
319         phase2 += phase;
320     }
321 
322     phase1 = 0;
323 
324     //Get phase at frequency 1.
325     for(i = 0; i < 100; i++)
326     {
327         Measure_4fs_CSFM(frequency1, 10);
328         phase1 += phase;
329     }
330 
331     //Average the collected phases.
332     phase2 /= 100;
333     phase1 /= 100;
334 
335     //Calculate phase difference.
336     deltaPhase = phase2 - phase1;
337 
338     //Calculate phase error slope.
339     calibrationPhaseSlope = deltaPhase / (frequency2 - frequency1);
340 
341     //Calculate the frequency where the slope intersects with phase = 0.
342     fOffset = (uint32_t)(phase2 / calibrationPhaseSlope);
343     calibrationZeroPhaseFrequency = frequency2 - fOffset;
344 }
345 
346 //Corrects the measured phase if calibration have been done.
347 double Measure_Calibrate_CorrectPhase(uint32_t measurementFrequency)
348 {
349     if(calibrationZeroPhaseFrequency == 0)
350     {
351         //No calibration has been done.
352         return 0;
353     }
354 
355     //Phase offset as calculated by calibration.
356     return ((double) (  ((double)calibrationZeroPhaseFrequency - (double)
        measurementFrequency) * calibrationPhaseSlope ));
```

```
357 }
358
359 //Calibrate system voltage.
360 void Measure_Calibrate_SystemVoltage(void)
361 {
362    uint8_t i;
363    uint16_t adcSampleValue[50];
364
365    //Initialize ADC to read internal voltage.
366    ADC1_InternalRef_Initialize();
367
368    //Enable ADC1.
369    ADC_Enable(ADC_1);
370
371    size_t numberOfElements = NUM_OF_ELEMENTS_IN_ARRAY(adcSampleValue);
372
373    Timer_Delay(10);
374
375    for(i = 0; i < numberOfElements; i++)
376    {
377       //Get single sample.
378       adcSampleValue[i] = ADC_GetSingleConversion(ADC_1);
379
380       adcSampleValue[i] = (uint16_t)((((double)TWELVE_BIT / (double)
              adcSampleValue[i]) * (double)INTERNAL_V_REF) * 1000);
381
382       //1 ms delay between samples.
383       Timer_Delay(1);
384    }
385
386    //Calculate system voltage from the last 48 samples of the total 50 taken
           . The first two samples are often way off.
387    Vcc_MilliVolt = DSP_AverageValueOfArray_ui16(&adcSampleValue[2],
           numberOfElements - 2);
388
389    ADC_Disable(ADC_1);
390    ADC_DisableInternalRef();
391 }
392
393 //Get system voltage.
394 uint16_t Measure_GetSystemVoltage(void)
395 {
396    return Vcc_MilliVolt;
397 }
398
399 //Get excitation voltage DC offset.
400 int16_t Measure_Excitation_GetDcOffset(void)
401 {
402    return exDcOffset;
403 }
404
405 //Set excitation voltage peak-to-peak in millivolts.
406 void Measure_SetExcitationPeakToPeakMillivolts(uint16_t exPtoPmV)
407 {
408    exPeakToPeakMillivolts = exPtoPmV;
409 }
410
411 //Get pointer to 4fs data array.
412 int16_t * Measure_4fs_GetDataArrayPointer(void)
413 {
414    return samplesPtr;
415 }
```

```
416
417 //Convenience function to set the temporary data array pointer.
418 int16_t * Measure_4fs_SetTempDataArrayPointer(int16_t * samplesPointer)
419 {
420    samplesTempPtr = samplesPointer;
421    return samplesTempPtr;
422 }
423
424 //Get the temporary data array pointer.
425 int16_t * Measure_4fs_GetTempDataArrayPointer()
426 {
427    return samplesTempPtr;
428 }
429
430 //Set the number of samples to sample.
431 void Measure_4fs_SetNumberOfSamples(uint16_t samplesToSet)
432 {
433    numberOfSamples = samplesToSet;
434 }
435
436 //Get the number of samples set.
437 uint16_t Measure_4fs_GetNumberOfSamples(void)
438 {
439    return numberOfSamples;
440 }
441
442 //Measure the excitation voltage.
443 void Measure_Excitation_UpdateParameters(uint32_t signalFrequency)
444 {
445    //Change DDS frequency if needed.
446    if(signalFrequency != DDS_GetFrequency())
447    {
448       DDS_Config(signalFrequency, 0, FALSE);
449    }
450
451    periods = MEASURE_EXCITATION_PERIODS;
452
453    //Calculate sampling timer parameters.
454    Measure_Calculate_SamplingParameters((uint32_t) (DDS_GetFrequencyFloat()
       + (float) 0.5), (uint32_t) ( (DDS_GetFrequencyFloat() *
       MEASURE_EXCITATION_SAMPLES_PER_PERIOD) + (float) 0.5), FALSE);
455
456    //Set sampling buffer.
457    DMA_SetSamplingBuffer(DMA_GetAdc3Struct(), &samples[0]);
458    DMA_SetTransferDone(DMA_GetAdc3Struct(), FALSE);
459
460    //Set number of samples to sample.
461    DMA_SetNumberOfSamples(DMA_GetAdc3Struct(), MEASURE_EXCITATION_PERIODS *
       MEASURE_EXCITATION_SAMPLES_PER_PERIOD);
462
463    DMA_ADC3_Initialize();
464
465    //Configure timer.
466    TIM_Disable(TIM_8);
467    TIM_SetPrescaler(TIM8_CH1_GetStruct(), timerPrescaler);
468    TIM_SetPeriod(TIM8_CH1_GetStruct(), timerPeriod);
469    TIM_SetClockDivider(TIM8_CH1_GetStruct(), TIM_CLK_DIV_1);
470    TIM_SetCounterMode(TIM8_CH1_GetStruct(), TIM_CNT_MODE_UP);
471    TIM8_CH1_Initialize();
472
473    //Enable DMA, ADC and Timer.
474    DMA_ADC3_Enable();
```

```
475     ADC_Enable(ADC_3);
476     TIM_Enable(TIM_8);
477 
478     while( DMA_IsTransferDone(DMA_GetAdc3Struct()) == FALSE );
479 
480     //Disable Timer, ADC and DMA.
481     TIM_Disable(TIM_8);
482     ADC_Disable(ADC_3);
483     DMA_ADC3_Disable();
484 
485     //Get the excitation DC offset.
486     exDcOffset = DSP_AverageValueOfArray_i16(&samples[0],
            DMA_GetNumberOfSamples(DMA_GetAdc3Struct()));
487 
488     //Remove the DC offset.
489     DSP_OffsetArray(&samples[0], DMA_GetNumberOfSamples(DMA_GetAdc3Struct()),
             exDcOffset);
490 
491     //Rectify sine wave
492     DSP_Rectify(&samples[0], DMA_GetNumberOfSamples(DMA_GetAdc3Struct()));
493 
494     //Average of rectified sine wave
495     exRectifiedAvg = DSP_AverageValueOfArray_i16_f(&samples[0],
            DMA_GetNumberOfSamples(DMA_GetAdc3Struct()));
496 
497     //RMS of sine wave
498     exRms = DSP_SineAverageToRms(exRectifiedAvg, DMA_GetNumberOfSamples(
            DMA_GetAdc3Struct()));
499 
500     //Average of rectified sine wave in millivolts
501     exRectifiedAvgMillivolts = Common_FromXBits((uint32_t)exRectifiedAvg, 12,
             (uint32_t)Measure_GetSystemVoltage());
502 
503     //RMS of sine wave converted to millivolts
504     exRmsMillivolts = Common_FromXBits((uint32_t)exRms, 12, (uint32_t)
            Measure_GetSystemVoltage());
505 
506     //RMS to Peak in millivolts (amplitude)
507     exAmplitudeMillivolts = DSP_RmsToSinePeak((float) exRmsMillivolts);
508 
509     //RMS to Peak-to-Peak in millivolts
510     exPeakToPeakMillivolts = exAmplitudeMillivolts * 2;
511 
512     DEBUG("DDS sine wave measurements:\nPeak to Peak (mV): %4.4f\nAmplitude (
            mV): %4.4f\nAverage (mV): %u\nRMS (mV): %i\n", exPeakToPeakMillivolts
            , exAmplitudeMillivolts, exRectifiedAvgMillivolts, exRmsMillivolts);
513 }
514 
515 //Calculates
516 void Measure_Calculate_IQPhaseAmplitude(bool_t calculate4Fs)
517 {
518     if(calculate4Fs)
519     {
520         //Calculate I and Q.
521         DSP_SamplesToIQ(Measure_4fs_GetTempDataArrayPointer(), &I[0], &Q[0],
                periods);
522 
523         //I and Q average
524         IAvg = DSP_AverageValueOfArray_i16(&I[0], periods);
525         QAvg = DSP_AverageValueOfArray_i16(&Q[0], periods);
526 
527         DEBUG("\nIavg: %i\nQavg: %i", IAvg, QAvg);
```

```
528
529         //When signal is very low close to noise floor, set I and Q average to
                0.
530         if( (abs(IAvg) < 10) && (abs(QAvg) < 10) )
531         {
532            IAvg = 0;
533            QAvg = 0;
534         }
535
536         //Calculate I and Q minimum and maximum values.
537         IMin = DSP_FindMin_i16(&I[0], periods);
538         QMin = DSP_FindMin_i16(&Q[0], periods);
539         IMax = DSP_FindMax_i16(&I[0], periods);
540         QMax = DSP_FindMax_i16(&Q[0], periods);
541
542         DEBUG("\nImin: %i\nImax: %i\nQmin: %i\nQmax: %i\nIdelta: %i\nQdelta: %
              i", IMin, IMax, QMin, QMax, IMax - IMin, QMax - QMin);
543
544         //IQ Amplitude and Phase
545         IQPeakToPeak = DSP_IQToAmplitude(IAvg, QAvg);
546         IQAmplitude = IQPeakToPeak / 2;
547         IQAmplitudeMilliVolts = Common_FromXBits((uint32_t)IQAmplitude, 12, (
              uint32_t)Measure_GetSystemVoltage());
548         IQPeakToPeakMilliVolts = IQAmplitudeMilliVolts * 2;
549
550         IQPhase = DSP_IQToPhase(IAvg, QAvg);
551         IQRmsMilliVolts = DSP_SinePeakToRms((double)IQAmplitudeMilliVolts);
552
553         DEBUG("\nPeak to Peak (mV): %u\nAmplitude (mV): %u\nRMS (mV): %4.4f\
              nPhase: %3.3f\nPhase Calibrated: %3.3f\nSignal Frequency: %7.3f",
              IQPeakToPeakMilliVolts, IQAmplitudeMilliVolts, IQRmsMilliVolts,
              IQPhase, IQPhaseCalibrated, DDS_GetFrequencyFloat());
554      }
555      else
556      {
557         //Calculate for normal
558      }
559   }
560
561   //Convenience function to calculate I to milliamperes.
562   double Immittance_Calculate_IToCurrentMilliAmpere(int16_t I)
563   {
564      iI = Immittance_Calculate_IQToCurrentMilliAmpere(I);
565      return iI;
566   }
567
568   //Convenience function to calculate Q to milliamperes.
569   double Immittance_Calculate_QToCurrentMilliAmpere(int16_t Q)
570   {
571      iQ = Immittance_Calculate_IQToCurrentMilliAmpere(Q);
572      return iQ;
573   }
574
575   //Calculate I or Q to milliampere.
576   double Immittance_Calculate_IQToCurrentMilliAmpere(int16_t IorQ)
577   {
578      double temp;
579
580      //ADC values to mV
581      temp = (double) Common_FromXBitsSigned(IorQ, 12, (uint32_t)
           Measure_GetSystemVoltage());
582
```

```
583     //Gain Stage 2.
584     temp /= (double)ELECTRIC_GAIN_STAGE_2_GAIN;
585
586     //Gain Stage 1.
587     temp /= (double)ELECTRIC_GAIN_STAGE_1_GAIN;
588
589     //Transimpedance amplifier stage.
590     temp /= (double)ELECTRIC_TRANSIMPEDANCE_GAIN;
591
592     return temp;
593 }
594
595 //Calculate the current peak-to-peak value from I and Q currents.
596 double Immittance_Calculate_CurrentPeakToPeakMilliAmpere(void)
597 {
598     //Current peak-to-peak = square_root( iI^2 + iQ^2 ).
599     iPeakToPeakMilliAmpere = sqrt((iI * iI) + (iQ * iQ));
600
601     return iPeakToPeakMilliAmpere;
602 }
603
604 //Gets the last calculated current peak-to-peak.
605 double Immittance_GetCurrentPeakToPeakMilliAmpere(void)
606 {
607     return iPeakToPeakMilliAmpere;
608 }
609
610 //Gets the last calculated current I component in milliampere.
611 double Immittance_GetCurrentIMilliAmpere(void)
612 {
613     return iI;
614 }
615
616 //Gets the last calculated current Q component in milliampere.
617 double Immittance_GetCurrentQMilliAmpere(void)
618 {
619     return iQ;
620 }
621
622 //Gets the last calculated current I component in microampere.
623 double Immittance_GetCurrentIMicroAmpere(void)
624 {
625     return (iI * 1000);
626 }
627
628 //Gets the last calculated current Q component in microampere.
629 double Immittance_GetCurrentQMicroAmpere(void)
630 {
631     return (iQ * 1000);
632 }
633
634 //Gets the last calculated current I component in nanoampere.
635 double Immittance_GetCurrentINanoAmpere(void)
636 {
637     return iI * 1000000;
638 }
639
640 //Gets the last calculated current Q component in nanoampere.
641 double Immittance_GetCurrentQNanoAmpere(void)
642 {
643     return iQ * 1000000;
644 }
```

```
645
646 //Calculates Impedance, Admittance, Resistance, Conductance, Reactance and
        Susceptance.
647 void Immittance_Calculate_ImmittanceParameters(void)
648 {
649    double dividend;
650    double phaseOffset;
651
652    //G : Conductance : [uS] = iI [nA] / U [mV]
653    G = Immittance_GetCurrentINanoAmpere() / (double)exPeakToPeakMillivolts;
654
655    //B : Susceptance : [uS] = iQ [nA] / U [mV]
656    B = Immittance_GetCurrentQNanoAmpere() / (double)exPeakToPeakMillivolts;
657
658    //Calculate dividend once.
659    dividend = ( (G * G) + (B * B) );
660
661    //R : Resistance  : [$\Omega$] = [S] / [S^2] = 1 / [S]
662    R = G / dividend;
663
664    //X : Reactance   : [X] = -[S] / [S^2] = 1 / [S]
665    X = -B / dividend;
666
667    //Scale resistance and reactance from micro Ohm to Ohm.
668    R *= 1000000;
669    X *= 1000000;
670
671    //Z : Impedance   : Z = R + jX
672    //|Z| : Modulus/Amplitude : |Z| = $sqrt( R^2 + X^2 )$.
673    Z = sqrtf( (R * R) + (X * X) );
674
675    //Y : Admittance  : Y = G + jB
676    //|Y| : Modulus/Amplitude : |Y| = $sqrt( G^2 + B^2 )$.
677    Y = sqrtf( (G * G) + (B * B) );
678
679    //Calculate phase.
680    phase = DSP_RadToDeg(atan2(X, R));
681
682    //Get phase to correct with.
683    phaseOffset = Measure_Calibrate_CorrectPhase(DDS_GetFrequency());
684
685    //Correct for calibrated phase offset.
686    R = Z*cos( DSP_DegToRad(phase + phaseOffset));
687    X = Z*sin( DSP_DegToRad(phase + phaseOffset));
688    Z = sqrtf( (R * R) + (X * X) );
689
690    //Calculate dividend once.
691    dividend = ( (R * R) + (X * X) );
692
693    //Admittance parameters in micro Siemens.
694    G = (R * 1000) / (dividend / 1000);
695    B = (-X * 1000) / (dividend / 1000);
696    Y = sqrtf( (G * G) + (B * B) );
697
698    //Update phase.
699    phase += phaseOffset;
700 }
701
702 //Gets the last calculated phase.
703 double Immittance_GetPhase(void)
704 {
705    return phase;
```

```
706 }
707
708 //Gets the last calculated impedance modulus.
709 double Immittance_GetImpedanceModulus(void)
710 {
711    return Z;
712 }
713
714 //Gets the last calculated admittance modulus in microsiemens.
715 double Immittance_GetAdmittanceModulusInMicroSiemens(void)
716 {
717    return Y;
718 }
719 //Gets the last calculated resistance.
720 double Immittance_GetResistance(void)
721 {
722    return R;
723 }
724
725 //Gets the last calculated reactance.
726 double Immittance_GetReactance(void)
727 {
728    return X;
729 }
730
731 //Gets the last calculated conductance in microsiemens.
732 double Immittance_GetConductanceInMicroSiemens(void)
733 {
734    return G;
735 }
736
737 //Gets the last calculated susceptance in microsiemens.
738 double Immittance_GetSusceptanceInMicroSiemens(void)
739 {
740    return B;
741 }
742
743 //Calculate all currents and immittance parameters from the current I and Q
        average values.
744 void Immittance_Start(void)
745 {
746    //Calculate current I component.
747    Immittance_Calculate_IToCurrentMilliAmpere(IAvg);
748
749    //Calculate current Q component.
750    Immittance_Calculate_QToCurrentMilliAmpere(QAvg);
751
752    //Calculate current peak-to-peak.
753    Immittance_Calculate_CurrentPeakToPeakMilliAmpere();
754
755    //Calculate immittance parameters.
756    Immittance_Calculate_ImmittanceParameters();
757 }
```

Listing A.51: MessageParser.h

```
#ifndef USER_ABSTRACT_MESSAGEPARSER_H_
#define USER_ABSTRACT_MESSAGEPARSER_H_

#include <stdint.h>

typedef struct MessageMeasurementSettings
{
   uint32_t realtimePlotEnable     : 1;
   uint32_t spectroscopyPlotEnable : 1;
   uint32_t realtimePlotParameters : 2;
   uint32_t technique              : 1;
   uint32_t frequency              : 20;
   uint32_t calibratePhase       : 1;
   uint32_t bmsMode            : 4;
   uint32_t reserved               : 3;
} MessageMeasurementSettings_t;

enum MessageIDIncoming
{
    PREAMBLE                        = 0xAA,
    MEASUREMENT_SETTINGS            = 0x01,

    MESSAGE_END                     = 0x0A
};

enum MessageEnumMeasurementSettings
{
    REALTIME_PLOT                       = 0,
    STATIC_PLOT                         = 1,
    ENABLE_PLOT                         = 1,
    DISABLE_PLOT                        = 0,
    CONTINUOUS_SINGLE_FREQUENCY_MODE    = 1,
    QUICK_FREQUENCY_SWEEP_MODE          = 2,
    FULL_FREQUENCY_SWEEP_MODE           = 3,
    RESISTANCE_AND_REACTANCE            = 1,
    CONDUCTANCE_AND_SUSEPTANCE          = 2,
    MODULUS_AND_PHASE                   = 3,
    TECHNIQUE_NORMAL                    = 0,
    TECHNIQUE_4FS                       = 1
};

void getAndParseNewMessages(void);

#endif /* USER_ABSTRACT_MESSAGEPARSER_H_ */
```

Listing A.52: MessageParser.c

```
1  #include <stdint.h>
2  #include "MessageParser.h"
3  #include "UART.h"
4  #include "Timer.h"
5  #include "Queues.h"
6  #include "Measurement.h"
7  #include "User.h"
8
9  //Parse measurement settings message.
10 static void parseMessageMeasurementSettings(uint8_t * data)
11 {
12    uint8_t i;
13    MessageMeasurementSettings_t msg = {0, 0, 0, 0, 0, 0, 0};
14    uint8_t * tempPtr = (uint8_t *) &msg;
15
16    for(i = 0; i < 4; i++)
17    {
18       tempPtr[i] = data[i];
19    }
20
21    //Update plot parameters.
22    Plot_UpdateParameters(&msg);
23 }
24
25 //Get UART data, check for messages and send them to the appropriate parsing
      function.
26 void getAndParseNewMessages(void)
27 {
28    uint32_t i = 0;
29    uint8_t temp = 0;
30    static uint8_t uncompleteMessageFromLastTime = 0;
31    uint8_t payload[10];
32
33    //Get UART data for 10 milliseconds.
34    USART1_EnableReceiveData();
35    Timer_Delay(10);
36    USART1_DisableReceiveData();
37
38    while(!Q8_isEmpty( USART1_GetReceiveQueue() ))
39    {
40       //If the buffer was empty last time trying to parse message, start
             with reading a byte from queue.
41       if(!uncompleteMessageFromLastTime)
42       {
43          temp = Q8_FifoRead(USART1_GetReceiveQueue());
44       }
45
46       //Check if start of message received.
47       if(temp == PREAMBLE || uncompleteMessageFromLastTime)
48       {
49          uncompleteMessageFromLastTime = 0;
50
51          //Check if there are enough data received to construct a message.
52          if(Q8_Count(USART1_GetReceiveQueue()) < 6)
53          {
54             //Break loop and try parse again next time.
55             uncompleteMessageFromLastTime = 1;
56             break;
57          }
58
```

```
59          //Get message ID.
60          temp = Q8_FifoRead(USART1_GetReceiveQueue());
61
62          //Handle received message.
63          switch (temp)
64          {
65             case MEASUREMENT_SETTINGS:
66                   //Get message payload data.
67                   for(i = 0; i < 4; i++)
68                   {
69                      payload[i] = Q8_FifoRead(USART1_GetReceiveQueue());
70                   }
71
72                   //Get end of message and check if valid.
73                   temp = Q8_FifoRead(USART1_GetReceiveQueue());
74                   if(temp == MESSAGE_END)
75                   {
76                      //End of message found, parse message.
77                      parseMessageMeasurementSettings(&payload[0]);
78                   }
79                break;
80
81             default:
82                //Do nothing, continue read from buffer and look for valid
                      messages.
83                break;
84          }
85       }
86    }
87 }
```

Listing A.53: Queues.h

```
#ifndef USER_QUEUES_H_
#define USER_QUEUES_H_

#include "defines.h"

//Structures
typedef struct
{
   volatile uint8_t buffer[QUEUE_UI8_SIZE];
   volatile uint16_t indexWrite;
   volatile uint16_t indexRead;
} Q8;

typedef struct
{
   uint16_t buffer[QUEUE_UI16_SIZE];
   uint16_t indexWrite;
   uint16_t indexRead;
} Q16;

typedef struct
{
   uint32_t buffer[QUEUE_UI32_SIZE];
   uint16_t indexWrite;
   uint16_t indexRead;
} Q32;

//Function prototypes
   //FIFO
void Q8_FifoWrite(Q8 * queue, uint8_t value);
uint8_t Q8_FifoRead(Q8 * queue);
uint8_t Q8_FifoReadElement(Q8 * queue, uint8_t element);
bool_t Q8_isEmpty(Q8 * queue);
uint16_t Q8_Count(Q8 * queue);

void Q16_FifoWrite(Q16 * queue, uint16_t value);
uint16_t Q16_FifoRead(Q16 * queue);
uint16_t Q16_FifoReadElement(Q16 * queue, uint16_t element);
bool_t Q16_isEmpty(Q16 * queue);
uint16_t Q16_Count(Q16 * queue);

void Q32_FifoWrite(Q32 * queue, uint32_t value);
uint32_t Q32_FifoRead(Q32 * queue);
uint32_t Q32_FifoReadElement(Q32 * queue, uint32_t element);
bool_t Q32_isEmpty(Q32 * queue);
uint16_t Q32_Count(Q32 * queue);

#endif /* USER_QUEUES_H_ */
```

Listing A.54: Queues.c

```
1  #include <stdint.h>
2  #include "defines.h"
3  #include "Queues.h"
4
5  //Write a byte to queue.
6  void Q8_FifoWrite(Q8 * queue, uint8_t value)
7  {
8     queue->buffer[ (queue->indexWrite++ & QUEUE_UI8_MASK) ] = value;
9  }
10
11 //Read/get next byte from queue.
12 //Must always check if FIFO is empty before doing a read.
13 uint8_t Q8_FifoRead(Q8 * queue)
14 {
15    return queue->buffer[ (queue->indexRead++ & QUEUE_UI8_MASK) ];
16 }
17
18 //Read the element oldest element in queue.
19 uint8_t Q8_FifoReadElement(Q8 * queue, uint8_t element)
20 {
21    return queue->buffer[ (queue->indexRead + element) ];
22 }
23
24 //Check if queue is empty.
25 bool_t Q8_isEmpty(Q8 * queue)
26 {
27    if(queue->indexRead == queue->indexWrite)
28    {
29       return TRUE;
30    }
31    else
32    {
33       return FALSE;
34    }
35 }
36
37 //Find how many elements in queue.
38 uint16_t Q8_Count(Q8 * queue)
39 {
40    return (queue->indexWrite - queue->indexRead);
41 }
42
43 //Write a half-word to queue.
44 void Q16_FifoWrite(Q16 * queue, uint16_t value)
45 {
46    queue->buffer[ (queue->indexWrite++ & QUEUE_UI16_MASK) ] = value;
47 }
48
49 //Read/get next half-word from queue.
50 //Must always check if FIFO is empty before doing a read.
51 uint16_t Q16_FifoRead(Q16 * queue)
52 {
53    return queue->buffer[ (queue->indexRead++ & QUEUE_UI16_MASK) ];
54 }
55
56 //Read the element oldest element in queue.
57 uint16_t Q16_FifoReadElement(Q16 * queue, uint16_t element)
58 {
59    return queue->buffer[ (queue->indexRead + element) ];
60 }
```

```
61
62  //Check if queue is empty.
63  bool_t Q16_isEmpty(Q16 * queue)
64  {
65     if(queue->indexRead == queue->indexWrite)
66     {
67        return TRUE;
68     }
69     else
70     {
71        return FALSE;
72     }
73  }
74
75  //Find how many elements in queue.
76  uint16_t Q16_Count(Q16 * queue)
77  {
78     return (queue->indexWrite - queue->indexRead);
79  }
80
81  //Write a word to queue.
82  void Q32_FifoWrite(Q32 * queue, uint32_t value)
83  {
84     queue->buffer[ (queue->indexWrite++ & QUEUE_UI16_MASK) ] = value;
85  }
86
87  //Read/get next word from queue.
88  //Must always check if FIFO is empty before doing a read.
89  uint32_t Q32_FifoRead(Q32 * queue)
90  {
91     return queue->buffer[ (queue->indexRead++ & QUEUE_UI16_MASK) ];
92  }
93
94  //Read the element oldest element in queue.
95  uint32_t Q32_FifoReadElement(Q32 * queue, uint32_t element)
96  {
97     return queue->buffer[ (queue->indexRead + element) ];
98  }
99
100 //Check if queue is empty.
101 bool_t Q32_isEmpty(Q32 * queue)
102 {
103    if(queue->indexRead == queue->indexWrite)
104    {
105       return TRUE;
106    }
107    else
108    {
109       return FALSE;
110    }
111 }
112
113 //Find how many elements in queue.
114 uint16_t Q32_Count(Q32 * queue)
115 {
116    return (queue->indexWrite - queue->indexRead);
117 }
```

Listing A.55: Timer.h

```
1  #ifndef USER_TIMER_H_
2  #define USER_TIMER_H_
3
4  void Timer_Start(uint32_t milliseconds);
5  bool_t Timer_isReached(void);
6  void Timer_Delay(uint32_t milliseconds);
7
8  void Timer_Test(void);
9
10 #endif /* USER_TIMER_H_ */
```

Listing A.56: Timer.c

```
1  #include "defines.h"
2  #include "LED.h"
3  #include "TIM.h"
4  #include "TIM2.h"
5  #include "SYSTICK.h"
6
7  static uint32_t startTick = 0;
8  static uint32_t timeToWait = 0;
9
10 //Start and set a time threshold.
11 void Timer_Start(uint32_t milliseconds)
12 {
13    //Start TIM2.
14    TIM_Enable(TIM_2);
15
16    //Get start tick.
17    startTick = TIM2_CH1_GetTick();
18
19    //Set minimum how many milliseconds to wait.
20    timeToWait = milliseconds;
21 }
22
23 //Checks if time threshold have been reached.
24 bool_t Timer_isReached(void)
25 {
26    uint32_t tick;
27    uint32_t deltaTime;
28
29    //Get current tick.
30    tick = TIM2_CH1_GetTick();
31
32    //Get time delta and check for counter wrap-around.
33    if(tick >= startTick)
34    {
35       deltaTime = tick - startTick;
36    }
37    else
38    {
39       deltaTime = (0xFFFFFFFF - startTick) + tick;
40    }
41
42    //Check if time threshold reached.
43    if (deltaTime >= timeToWait)
44    {
45       //Stop TIM2.
46       TIM_Disable(TIM_2);
47       return TRUE;
48    }
49    else
50    {
51       return FALSE;
52    }
53 }
54
55 //Blocking delay
56 void Timer_Delay(uint32_t milliseconds)
57 {
58    SYSTICK_Delay(milliseconds);
59 }
60
```

```
61 //LED should blink once every second.
62 void Timer_Test(void)
63 {
64    while(TRUE)
65    {
66       //LED(LED1, TOGGLE);
67       Timer_Start(1000);
68       while(!Timer_isReached());
69    }
70 }
```

Listing A.57: User.h

```
#ifndef USER_USER_H_
#define USER_USER_H_

void User_SaveAllSettings(void);

//Update plot parameters.
void Plot_UpdateParameters(MessageMeasurementSettings_t * msg);
MessageMeasurementSettings_t * Plot_GetRealtime(void);
MessageMeasurementSettings_t * Plot_GetStatic(void);
MessageMeasurementSettings_t * Calibration_GetMessage(void);
MessageMeasurementSettings_t * User_GetBmsModeMsg(void);
void Calibration_DonePhase(void);
void BMS_IdleMode(void);

#endif /* USER_USER_H_ */
```

Listing A.58: User.c

```
#include "MessageParser.h"
#include "User.h"
#include "FLASH.h"
#include "BMS.h"

//Plot parameters instances.
static MessageMeasurementSettings_t realtimePlot = {0,0,0,0,0,0,0,0};
static MessageMeasurementSettings_t staticPlot = {0,0,0,0,0,0,0,0};
static MessageMeasurementSettings_t calibration = {0,0,0,0,0,0,0,0};
static MessageMeasurementSettings_t bmsModeMsg = {0,0,0,0,0,0,0,0};

//Save settings to flash memory.
void User_SaveAllSettings(void)
{

}

//
void Plot_UpdateParameters(MessageMeasurementSettings_t * msg)
{
   realtimePlot.realtimePlotEnable = msg->realtimePlotEnable;
   realtimePlot.realtimePlotParameters = msg->realtimePlotParameters;
   realtimePlot.frequency = msg->frequency;
   realtimePlot.technique = msg->technique;

   staticPlot.spectroscopyPlotEnable = msg->spectroscopyPlotEnable;
   staticPlot.technique = msg->technique;

   calibration.calibratePhase = msg->calibratePhase;

   bmsModeMsg.bmsMode = msg->bmsMode;
}

MessageMeasurementSettings_t * Plot_GetRealtime(void)
{
   return &realtimePlot;
}

MessageMeasurementSettings_t * Plot_GetStatic(void)
{
   return &staticPlot;
}

MessageMeasurementSettings_t * Calibration_GetMessage(void)
{
   return &calibration;
}

MessageMeasurementSettings_t * User_GetBmsModeMsg(void)
{
   return &bmsModeMsg;
}

void Calibration_DonePhase(void)
{
   calibration.calibratePhase = 0;
}

void BMS_IdleMode(void)
{
```

```
61      bmsModeMsg.bmsMode = IDLE;
62  }
```

Listing A.59: BMS.h

```
#ifndef USER_BMS_H_
#define USER_BMS_H_

typedef enum
{
   STARTUP = 0,
   IDLE,
   MEASURE,
   EGG_MODE,

   //States below this line should generally not be used as arguments for
       functions, except from within SystemMode function.
   _STARTUP = 10000,
   _IDLE,
   _MEASURE,
   _EGG_MODE
} BMS_MODE_t;

void BMS(void);
void BMS_SetMode(BMS_MODE_t systemMode);
BMS_MODE_t BMS_GetMode();
BMS_MODE_t BMS_GetPreviousMode();

//Modes functions.
void BMS_Startup(void);
void BMS_ModeMeasure(void);
void BMS_ModeEgg(void);

//On enter/transition modes functions.
void BMS_OnEnter_MeasureMode(void);
void BMS_OnEnter_EggMode(void);

//Other
void BMS_HeartBeat(void);
void BMS_CalibrationStatus(void);

#endif /* USER_BMS_H_ */
```

Listing A.60: BMS.c

```
1  #include <math.h>
2  #include "BMS.h"
3  #include "defines.h"
4  #include "Initialize.h"
5  #include "MessageParser.h"
6  #include "Measurement.h"
7  #include "Timer.h"
8  #include "ADC.h"
9  #include "ADC1.h"
10 #include "ADC2.h"
11 #include "ADC3.h"
12 #include "DDS.h"
13 #include "IWDG.h"
14 #include "SYSTICK.h"
15 #include "User.h"
16 #include "ZCD.h"
17
18 static BMS_MODE_t currentBMSMode = STARTUP;
19 static BMS_MODE_t previousBMSMode1 = IDLE;
20 static BMS_MODE_t previousBMSMode2 = IDLE;
21 static bool_t calibrationDone = FALSE;
22
23 //Array of frequencies with equal spacing in a logarithmic scale, used in
       spectroscopy mode.
24 unsigned int spectroscopyFrequencies[88] = {1000, 1058, 1120, 1186, 1255,
       1328, 1406, 1488, 1575, 1667, 1765, 1868, 1977, 2093, 2215, 2344, 2481,
       2626, 2780, 2942, 3114, 3296, 3489, 3693, 3909, 4137, 4379, 4635, 4906,
       5192, 5496, 5817, 6157, 6517, 6898, 7301, 7728, 8179, 8657, 9163, 9699,
       10266, 10866, 11501, 12173, 12884, 13638, 14435, 15278, 16171, 17116,
       18117, 19175, 20296, 21482, 22738, 24067, 25473, 26962, 28538, 30206,
       31971, 33840, 35817, 37911, 40126, 42472, 44954, 47581, 50362, 53305,
       56421, 59718, 63208, 66903, 70813, 74951, 79332, 83968, 88876, 94070,
       99568, 105387, 111547, 118066, 124966, 132270, 140000};
25
26 //In charge of executing the current system state, sending a heartbeat
       signal
27 //over UART, get messages and update the watchdog.
28 void BMS(void)
29 {
30    MessageMeasurementSettings_t * BMSModeReceived;
31    static uint32_t counter = 0;
32
33    BMSModeReceived = User_GetBmsModeMsg();
34
35    if(BMSModeReceived->bmsMode != BMS_GetMode())
36    {
37       BMS_SetMode(BMSModeReceived->bmsMode);
38    }
39
40    switch (currentBMSMode)
41    {
42       case STARTUP:
43          BMS_Startup();
44
45          //Go to IDLE mode.
46          BMS_SetMode(IDLE);
47          BMS_IdleMode();
48          break;
49
50       case IDLE:
```

```
51          //Do nothing.
52          break;
53
54       case MEASURE:
55          BMS_OnEnter_MeasureMode();
56          BMS_SetMode(_MEASURE);
57          break;
58
59       case _MEASURE:
60          BMS_ModeMeasure();
61          break;
62
63       case EGG_MODE:
64          BMS_OnEnter_EggMode();
65          BMS_SetMode(_EGG_MODE);
66          break;
67
68       case _EGG_MODE:
69          BMS_ModeEgg();
70          break;
71
72       default:
73          BMS_SetMode(IDLE);
74          break;
75    }
76
77    //Do this every 50th iteration.
78    if( (counter % 50) == 0)
79    {
80       //Send heart beat signal to GUI.
81       BMS_HeartBeat();
82
83       //Send calibration status.
84       BMS_CalibrationStatus();
85    }
86
87    //Get and parse incomming data.
88    getAndParseNewMessages();
89
90    //Update Watchdog.
91    IWDG_Update();
92
93    //Increment loop counter.
94    counter++;
95 }
96
97 //Set BMS mode.
98 void BMS_SetMode(BMS_MODE_t bmsMode)
99 {
100    previousBMSMode2 = previousBMSMode1;
101    previousBMSMode1 = currentBMSMode;
102    currentBMSMode = bmsMode;
103 }
104
105 //Get current BMS mode.
106 BMS_MODE_t BMS_GetMode()
107 {
108    if(currentBMSMode >= 10000)
109    {
110       return (currentBMSMode - 10000);
111    }
112
```

```
113     return currentBMSMode;
114 }
115 
116 //Get previous BMS mode.
117 BMS_MODE_t BMS_GetPreviousMode()
118 {
119     //Get previous BMS mode, but skip transition states.
120     if(!(previousBMSMode1 >= 10000))
121     {
122         return previousBMSMode1;
123     }
124     else
125     {
126         return previousBMSMode2;
127     }
128 }
129 
130 //Start-up mode.
131 void BMS_Startup(void)
132 {
133     //Initializations.
134     Initialize();
135 
136     //Enable global interrupts.
137     __enable_irq();
138 }
139 
140 //Transition to measure mode.
141 void BMS_OnEnter_MeasureMode(void)
142 {
143     Measure_Calibrate_SystemVoltage();
144     ADC1_4fs_Initialize();
145     ZCD_Initialize();
146     Measure_SetExcitationPeakToPeakMillivolts(100);
147 }
148 
149 //Measure mode.
150 void BMS_ModeMeasure(void)
151 {
152     MessageMeasurementSettings_t * realtimePlot;
153     MessageMeasurementSettings_t * staticPlot;
154     MessageMeasurementSettings_t * calibration;
155 
156     unsigned int i;
157     realtimePlot = Plot_GetRealtime();
158     staticPlot = Plot_GetStatic();
159     calibration = Calibration_GetMessage();
160     double rTemp;
161     double xTemp;
162 
163     //Update real-time parameters.
164     if(realtimePlot->realtimePlotEnable)
165     {
166         if(realtimePlot->technique == TECHNIQUE_4FS)
167         {
168             rTemp = 0;
169             xTemp = 0;
170 
171             for(i = 0; i < 10; i++)
172             {
173                 Measure_4fs_CSFM(realtimePlot->frequency, 10);
174                 rTemp += Immittance_GetResistance();
```

```
175             xTemp += Immittance_GetReactance();
176         }
177
178         //Get average.
179         rTemp /= 10;
180         xTemp /= 10;
181
182         SEND_MSG_RESISTANCE_REACTANCE(rTemp, xTemp);
183         SEND_MSG_CSFM_FREQUENCY((unsigned int)DDS_GetFrequency());
184       }
185       else if(realtimePlot->technique == TECHNIQUE_NORMAL)
186       {
187         //Normal mode.
188       }
189     }
190
191     //Update spectroscopy parameters.
192     if(staticPlot->spectroscopyPlotEnable)
193     {
194       if(staticPlot->technique == TECHNIQUE_4FS)
195       {
196         //Do frequency sweep.
197         Measure_4fs_Spectroscopy(&spectroscopyFrequencies[0],
              NUM_OF_ELEMENTS_IN_ARRAY(spectroscopyFrequencies), 10);
198       }
199       else if(staticPlot->technique == TECHNIQUE_NORMAL)
200       {
201         //Do frequency sweep.
202
203       }
204     }
205
206     //If both plots are disabled, power down DDS.
207     if(!realtimePlot->realtimePlotEnable && !staticPlot->
          spectroscopyPlotEnable)
208     {
209       //Power down DDS.
210       DDS_Config(0,0,TRUE);
211     }
212
213     //If calibration requested and calibration not done, do it.
214     if(calibration->calibratePhase && !calibrationDone)
215     {
216       Measure_Calibrate_CalculatePhaseErrorSlope(10000, 100000);
217       Calibration_DonePhase();
218       calibrationDone = TRUE;
219     }
220 }
221
222 //Transition to egg demo mode.
223 void BMS_OnEnter_EggMode(void)
224 {
225     //Calibrate system voltage.
226     Measure_Calibrate_SystemVoltage();
227
228     //Initialize ADC1.
229     ADC1_4fs_Initialize();
230
231     //Initialize Zero Crossing Detector.
232     ZCD_Initialize();
233
234     //Set excitation voltage.
```

```
235     Measure_SetExcitationPeakToPeakMillivolts(100);
236 }
237
238 //Egg demo mode.
239 void BMS_ModeEgg(void)
240 {
241     unsigned int i = 0;
242     double phase3k;
243     double phase30k;
244     uint32_t mod3k;
245     uint32_t mod30k;
246     double deltaPhase;
247
248     phase3k = 0;
249     mod3k = 0;
250     phase30k = 0;
251     mod30k = 0;
252
253     for(i = 0; i < 10; i++)
254     {
255         //Measure at 3 kHz.
256         Measure_4fs_CSFM(3000,10);
257
258         //Get impedance modulus and phase.
259         phase3k += Immittance_GetPhase();
260         mod3k += (uint32_t)Immittance_GetImpedanceModulus();
261     }
262
263     for(i = 0; i < 10; i++)
264     {
265         //Measure at 30 kHz.
266         Measure_4fs_CSFM(30000,10);
267
268         //Get impedance modulus and phase.
269         phase30k += Immittance_GetPhase();
270         mod30k += (uint32_t)Immittance_GetImpedanceModulus();
271     }
272
273     //Calculate average.
274     phase3k /= 10;
275     mod3k /= 10;
276     phase30k /= 10;
277     mod30k /= 10;
278
279     //Calculate phase difference.
280     deltaPhase = fabs(phase3k - phase30k);
281
282     //Check if needle is in egg yet.
283     if( (mod3k > 5000) && (mod30k > 5000) )
284     {
285         SEND_MSG_EGG_STATE(0);
286         return;
287     }
288     else if( ( mod3k < 100 ) || (mod30k < 100) )
289     {
290         SEND_MSG_EGG_STATE(0);
291         return;
292     }
293
294     //Determine where in the egg the needle is at.
295     if(deltaPhase > 32)
296     {
```

```
297         //Egg white.
298         SEND_MSG_EGG_STATE(1);
299 
300     }
301     else if(deltaPhase < 28)
302     {
303         //Egg yolk.
304         SEND_MSG_EGG_STATE(2);
305     }
306     else
307     {
308         //In between.
309         SEND_MSG_EGG_STATE(3);
310     }
311 }
312 
313 //Sends a heartbeat signal.
314 void BMS_HeartBeat(void)
315 {
316     SEND_MSG_HEARTBEAT;
317 }
318 
319 //Sends the calibration status of the system.
320 void BMS_CalibrationStatus(void)
321 {
322     SEND_MSG_CALIBRATION_STATUS((unsigned int) calibrationDone);
323 }
```

Listing A.61: ISR.c

```
1  #include "stm32f10x_adc.h"
2  #include "stm32f10x_dma.h"
3  #include "stm32f10x_exti.h"
4  #include "stm32f10x_tim.h"
5  #include "stm32f10x_usart.h"
6  #include "defines.h"
7  #include "Queues.h"
8  #include "ADC.h"
9  #include "ADC2.h"
10 #include "DMA.h"
11 #include "DSP.h"
12 #include "LED.h"
13 #include "LSI.h"
14 #include "SYSTICK.h"
15 #include "TIM.h"
16 #include "TIM1.h"
17 #include "TIM2.h"
18 #include "TIM8.h"
19 #include "UART.h"
20
21 void NMI_Handler(void)
22 {
23 }
24
25 void HardFault_Handler(void)
26 {
27    while(TRUE);
28 }
29
30 void MemManage_Handler(void)
31 {
32    while(TRUE);
33 }
34
35 void BusFault_Handler(void)
36 {
37    while(TRUE);
38 }
39
40 void UsageFault_Handler(void)
41 {
42    while(TRUE);
43 }
44
45 void SVC_Handler(void)
46 {
47 }
48
49 void DebugMon_Handler(void)
50 {
51 }
52
53 void PendSV_Handler(void)
54 {
55 }
56
57 //This function handles Zero Crossing Detector interrupt request.
58 void EXTI2_IRQHandler(void)
59 {
60    if( (EXTI->PR & EXTI_Line2) != RESET)
```

```
61      {
62          //Enable Timer 1
63          TIM1->CR1 |= TIM_CR1_CEN;
64
65          //Disable zero crossing interrupt.
66          EXTI->IMR &= ZERO_CROSSING_DETECTOR_DISABLE_INTERRUPT_MASK;
67
68          //Clear interrupt.
69          EXTI->PR = EXTI_Line2;
70      }
71  }
72
73  //IQ DMA ISR
74  void DMA1_Channel1_IRQHandler(void)
75  {
76      if((DMA1->ISR & DMA_ISR_TCIF1) != RESET)
77      {
78          //Disable TIM1.
79          TIM1->CR1 &= (~TIM_CR1_CEN);
80
81          //Disable ADC1.
82          ADC1->CR2 &= ((uint32_t)0xFFFFFFFE);
83
84          //Set DMA flags.
85          DMA_SetTransferDone(DMA_Get4fsStruct(), TRUE);
86          DMA_SetTransferDone(DMA_GetNormalStruct(), TRUE);
87
88          //Clear DMA transfer complete interrupt flag.
89          DMA1->IFCR = ~DMA_IFCR_CTCIF1;
90      }
91
92      if((DMA1->ISR & DMA_ISR_TEIF1) != RESET)
93      {
94          //Clear DMA transfer error interrupt flag.
95          DMA1->IFCR = ~DMA_IFCR_CTEIF1;
96      }
97  }
98
99  //DDS DMA ISR
100 void DMA2_Channel4_5_IRQHandler(void)
101 {
102     if(DMA_GetITStatus(DMA2_IT_TC5) != RESET)
103     {
104         //Set DMA flag.
105         DMA_SetTransferDone(DMA_GetAdc3Struct(), TRUE);
106
107         //Clear DMA transfer complete interrupt flag
108         DMA_ClearITPendingBit(DMA2_IT_TC5);
109     }
110
111     if(DMA_GetITStatus(DMA2_IT_TE5) != RESET)
112     {
113         //Clear DMA transfer error interrupt flag.
114         DMA_ClearITPendingBit(DMA2_IT_TE5);
115     }
116 }
117
118 //System tick ISR.
119 void SysTick_Handler(void)
120 {
121     SYSTICK_DecrementCounter();
122 }
```

```
123
124 //Timer 1 CC interrupt.
125 //ADC1 sampling timer (4fs or Normal).
126 void TIM1_CC_IRQHandler(void)
127 {
128    if( (TIM1->SR & TIM_IT_CC1) != RESET)
129    {
130       //Clear Timer 1 CC1 interrupt.
131       TIM1->SR = ~TIM_IT_CC1;
132    }
133 }
134
135 //Millisecond precision hardware counter.
136 void TIM2_IRQHandler(void)
137 {
138    if(TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
139    {
140       //Increment millisecond counter.
141       timer2Counter++;
142
143       //Clear Timer 2 CC1 interrupt.
144       TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
145    }
146 }
147
148 //Calibrate LSI ISR.
149 void TIM5_IRQHandler(void)
150 {
151    static uint16_t timeStamp1;
152    uint16_t timeStamp2;
153    uint32_t timeDelta;
154    uint32_t temp;
155
156    if(TIM_GetITStatus(TIM5, TIM_IT_CC4) != RESET)
157    {
158       if(LSI_GetCounterValue() == 0)
159       {
160          //Get input capture value.
161          timeStamp1 = TIM_GetCapture4(TIM5);
162       }
163       else if(LSI_GetCounterValue() == 1)
164       {
165          //Get input capture value.
166          timeStamp2 = TIM_GetCapture4(TIM5);
167
168          //Get time delta and check for counter wrap-around.
169          if(timeStamp2 > timeStamp1)
170          {
171             timeDelta = (timeStamp2 - timeStamp1);
172          }
173          else
174          {
175             timeDelta = ((0xFFFF - timeStamp1) + timeStamp2);
176          }
177
178          //Compute LSI frequency.
179          temp = (uint32_t) (SYSTEM_FREQUENCY / timeDelta);
180
181          //Compensate for Timer 5 input compare prescaler.
182          temp *= 8;
183
184          //Set LSI frequency.
```

```
185             LSI_SetFrequency(temp);
186         }
187
188         LSI_SetCounterValue(1);
189
190         //Clear Timer 5 CC4 interrupt.
191         TIM_ClearITPendingBit(TIM5, TIM_IT_CC4);
192     }
193 }
194
195 //Timer 8 CC interrupt.
196 //ADC3 sampling timer (DDS).
197 void TIM8_CC_IRQHandler(void)
198 {
199     if(TIM_GetITStatus(TIM8, TIM_IT_CC1) != RESET)
200     {
201         //Clear Timer 8 CC1 interrupt.
202         TIM_ClearITPendingBit(TIM8, TIM_IT_CC1);
203     }
204 }
205
206 //USART1 ISR.
207 void USART1_IRQHandler(void)
208 {
209     //Check if a receive interrupt was triggered.
210     if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
211     {
212         //Get byte and write to queue.
213         Q8_FifoWrite(USART1_GetReceiveQueue(), USART_ReceiveData(USART1));
214
215         //Clear USART1 receive data interrupt flag.
216         USART_ClearITPendingBit(USART1, USART_IT_RXNE);
217     }
218 }
```

Listing A.62: FIRcoef.h

```
/*
 * Discrete-Time FIR Filter (real)
 * -------------------------------
 * Filter Structure  : Direct-Form FIR
 * Filter Length     : 16
 * Stable            : Yes
 * Linear Phase      : Yes (Type 2)
 * Arithmetic        : fixed
 * Numerator         : s32,19 -> [-4096 4096)
 * Input             : s16,15 -> [-1 1)
 * Filter Internals  : Full Precision
 *   Output          : s38,34 -> [-8 8)  (auto determined)
 *   Product         : s36,34 -> [-2 2)  (auto determined)
 *   Accumulator     : s38,34 -> [-8 8)  (auto determined)
 *   Round Mode      : No rounding
 *   Overflow Mode   : No overflow
 */

/* General type conversion for MATLAB generated C-code  */
//#include "tmwtypes.h"
//#include <stdint.h>

//const int FIR_coefLength = 16;
const q31_t FIR_coef2[16] = {
          3649,       18509,       56054,      126123,      229014,
             349606,
        458883,      524288,      524288,      458883,      349606,
           229014,
        126123,       56054,       18509,        3649
};

q31_t FIR_coef[13] = {
      -5689815,    31046402,   143604095,   353776472,   621756431,
          852259293,
     943654557,   852259293,   621756431,   353776472,   143604095,
         31046402,
      -5689815
};

q31_t FIR_input[1885] = {
      64, 87, 42, 106, 60, 83, 105, 59, 121, 74, 94, 114, 66, 126, 76, 94,
          112, 61, 119, 67, 83, 98, 46, 102, 49, 64, 79, 26, 82, 30, 45, 61,
           9, 67, 16, 34, 52, 2, 62, 14, 34, 54, 7, 69, 23, 45, 68, 22, 86,
          41, 64, 87, 42, 106, 60, 83, 105, 59, 121, 74, 94, 114, 66, 126,
          76, 94, 112, 61, 119, 67, 83, 98, 46, 102, 49, 64, 79, 26, 82, 30,
           45, 61, 9, 67, 16, 34, 52, 2, 62, 14, 34, 54, 7, 69, 23, 45, 68,
          22, 86, 41, 64, 87, 42, 106, 60, 83, 105, 59, 121, 74, 94, 114,
          66, 126, 76, 94, 112, 61, 119, 67, 83, 98, 46, 102, 49, 64, 79,
          26, 82, 30, 45, 61, 9, 67, 16, 34, 52, 2, 62, 14, 34, 54, 7, 69,
          23, 45, 68, 22, 86, 41, 64, 87, 42, 106, 60, 83, 105, 59, 121, 74,
           94, 114, 66, 126, 76, 94, 112, 61, 119, 67, 83, 98, 46, 102, 49,
          64, 79, 26, 82, 30, 45, 61, 9, 67, 16, 34, 52, 2, 62, 14, 34, 54,
          7, 69, 23, 45, 68, 22, 86, 41, 64, 87, 42, 106, 60, 83, 105, 59,
          121, 74, 94, 114, 66, 126, 76, 94, 112, 61, 119, 67, 83, 98, 46,
          102, 49, 64, 79, 26, 82, 30, 45, 61, 9, 67, 16, 34, 52, 2, 62, 14,
           34, 54, 7, 69, 23, 45, 68, 22, 86, 41, 64
};
```

Listing A.63: defines.h

```
#ifndef USER_DEFINES_H_
#define USER_DEFINES_H_

#include <stdio.h>
#include <stdint.h>
#include <stm32f10x.h>
#include <stm32f10x_gpio.h>

//Enable/disable options.
#define SEND_PRINT
//#define DEBUG_PRINT
//#define DEBUG_MODE

//Common.
typedef enum { FALSE = 0, TRUE } bool_t;
#define OK                                          0
#define FAIL                                        -1
#define FALSE                                       0
#define TRUE                                        1
#define TWELVE_BIT                                     4096
#define NUM_OF_ELEMENTS_IN_ARRAY(x)                       (sizeof(x)/sizeof(x
    [0]))

//Output modes.
#ifdef SEND_PRINT

#define SEND(fmt, ...)
    printf(fmt "\n",##__VA_ARGS__)
#define SEND_MSG_DDS(excitationAverage)
       printf( "1:%i\n", excitationAverage)
#define SEND_MSG_AMPLITUDE_PHASE(IQRmsMilliVolts, IQPhase)
            printf( "2:%4.4f:%3.3f\n", IQRmsMilliVolts, IQPhase)
#define SEND_CURRENT(iPeakToPeakMilliAmpere)
          printf( "9:%3.5f\n", iPeakToPeakMilliAmpere)
#define SEND_MSG_RESISTANCE_REACTANCE(resistance, reactance)
            printf( "10:%7.0f:%7.0f\n", resistance, reactance)
#define SEND_MSG_SPECTROSCOPY_RESISTANCE_REACTANCE(frequency, resistance,
    reactance)      printf( "13:%u:%7.0f:%7.0f\n", frequency, resistance,
    reactance)
#define SEND_MSG_CSFM_FREQUENCY(frequency)
         printf( "14:%u\n", frequency)
#define SEND_MSG_CONDUCTANCE_SUSCEPTANCE(conductance, susceptance)
               printf( "11:%7.0f:%7.0f\n", conductance, susceptance)
#define SEND_MSG_IMPEDANCE_ADMITTANCE(impedance, admittance)
            printf( "12:%7.0f:%7.0f\n", impedance, admittance)
#define SEND_MSG_EGG_STATE(state)
       printf( "15:%u\n", state)
#define SEND_MSG_HEARTBEAT
    printf( "99:0\n");
#define SEND_MSG_CALIBRATION_STATUS(state)
          printf( "98:%u\n", state)

#else
#define SEND(fmt, ...)
#define SEND_MSG_DDS(excitationAverage)
#define SEND_MSG_AMPLITUDE_PHASE(IQRmsMilliVolts, IQPhase)
#define SEND_CURRENT(iPeakToPeakMilliAmpere)
#endif

#ifdef DEBUG_PRINT
```

```
47 #define DEBUG(fmt, ...)                                printf(fmt "\n",##
       __VA_ARGS__)
48 #else
49 #define DEBUG(fmt, ...)
50 #endif
51
52 //Power
53 #define INTERNAL_V_REF                                 1.2f
54
55 //Clocks.
56 #define SYSTEM_FREQUENCY                               ((uint32_t)72000000)
57
58 //USART1.
59 #define USART1_BAUDRATE                                115200
60 #define USART1_RX_BUFFER_SIZE                          256
61
62 //Flash.
63 #define FLASH_BASE_ADDRESS                             ((uint32_t) 0x08000000)
64 #define FLASH_PAGE_SIZE                                ((uint16_t) 0x800)
65 #define FLASH_STORAGE_START_ADDRESS                       ((uint32_t) 0x08060000
       )
66 #define PAGES_TO_ERASE                                 2
67 #define LAST_PAGE_START_ADDRESS                           ((uint32_t) 0x0807F800
       )
68 #define FLASH_TOTAL_PAGES                              ((LAST_PAGE_START_ADDRESS
        + FLASH_PAGE_SIZE - FLASH_STORAGE_START_ADDRESS) / FLASH_PAGE_SIZE)
69
70 //DDS.
71 #define DDS_PORT_D7                                    GPIOE
72 #define DDS_PORT_WCLK                                  GPIOE
73 #define DDS_PORT_FQUD                                  GPIOB
74 #define DDS_PORT_RESET                                 GPIOB
75
76 #define DDS_D7                                         GPIO_Pin_12
77 #define DDS_WCLK                                    GPIO_Pin_14
78 #define DDS_FQUD                                    GPIO_Pin_10
79 #define DDS_RESET                                      GPIO_Pin_11
80
81 #define DDS_CLOCK_RESOLUTION
       0.029103830456733703613281250f
82 #define DDS_MAX_FREQ_WORD_SIZE                            (0xFFFFFFFF / (1 /
       DDS_CLOCK_RESOLUTION))
83
84 //FIR filter.
85 #define FIR_FILTER_COEF_LENGTH                            13
86 #define FIR_FILTER_SAMPLES                                NUMBER_OF_SAMPLES
87
88 //DSP.
89 #define V_AVG_TO_RMS                                   1.1107f     // Vrms = (pi
       /2*sqrt(2)) * Vavg = 1.1107
90
91    //Signal.
92 #define NUMBER_OF_PERIODS                              20
93 #define SAMPLING_FACTOR                                32
94 #define NUMBER_OF_SAMPLES                              (SAMPLING_FACTOR *
       NUMBER_OF_PERIODS) + 1
95
96    //Reference.
97 #define SHIFT_REFERENCE_BY_SAMPLES                        16
98 #define REF_DATA_ANALOG_TO_DIGITAL_THRESHOLD              (1 << ADC1_RESOLUTION)
       /2
99 #define REF_DUTY_CYCLE_LOWER_THRESHOLD                    0.45f
```

```
100 #define REF_DUTY_CYCLE_UPPER_THRESHOLD                0.55f
101
102    //Filter.
103 #define BLOCK_SIZE                                  1
104 #define FILTER_ORDER                               32
105 #define NUMBER_OF_TAPS                              (FILTER_ORDER + 1)
106
107 //DSP2.
108 #define MOVING_AVERAGE_BUFFER_SIZE                    16
109 #define MOVING_AVERAGE_BUFFER_SIZE_MASK                  (
        MOVING_AVERAGE_BUFFER_SIZE - 1)
110 #define ZERO_CROSSING_DETECTOR_ENABLE_INTERRUPT_MASK     0x00000004
111 #define ZERO_CROSSING_DETECTOR_DISABLE_INTERRUPT_MASK    0x0007FFFB
112
113 //ADC.
114 #define ADC_MAX_SAMPLING_SPEED                        800000
115
116 //ADC1.
117 #define ADC1_RESOLUTION                             12
118 #define ADC1_DATA_REGISTER_ADDRESS                     0x4001244C
119 #define ADC_DATA_BUFFER_SIZE                        180
120
121 //ADC3 (DDS amplitude).
122 #define ADC3_DATA_REGISTER_ADDRESS                    0x40013C4C
123
124 //ADC 4fs debug.
125 #define DEBUG_ADC_INPUT_WAVEFORM_SAMPLES              16 //This number must
        equal 2^n. The number of samples per period for debug is then 4 *
        DEBUG_ADC_INPUT_WAVEFORM_SAMPLES.
126
127 uint32_t ADCdata[NUMBER_OF_SAMPLES];
128 int16_t ADCdataSignal[NUMBER_OF_SAMPLES];
129 int16_t ADCdataRef[NUMBER_OF_SAMPLES];
130
131 //Queues.
132 #define QUEUE_UI8_SIZE                              1024
133 #define QUEUE_UI8_MASK                              (QUEUE_UI8_SIZE - 1)
134 #define QUEUE_UI16_SIZE                             64
135 #define QUEUE_UI16_MASK                             (QUEUE_UI16_SIZE - 1)
136 #define QUEUE_UI32_SIZE                             64
137 #define QUEUE_UI32_MASK                             (QUEUE_UI32_SIZE - 1)
138
139 //Measurement.
140 #define MEASURE_SAMPLE_ARRAY_SIZE                     8192
141 #define MEASURE_MAX_SAMPLING_PERIODS                  255
142 #define MEASURE_4FS_RATIO                           4
143 #define MEASURE_EXCITATION_PERIODS                    32
144 #define MEASURE_EXCITATION_SAMPLES_PER_PERIOD            64
145
146 //Electrical.
147 #define ELECTRIC_EXCITATION_PEAK_TO_PEAK_MILLIVOLT       100
148 #define ELECTRIC_TRANSIMPEDANCE_GAIN                  10789
149 #define ELECTRIC_GAIN_STAGE_1_GAIN                    1.73360737919f
150 #define ELECTRIC_GAIN_STAGE_2_GAIN                    0.7797f
151
152 #endif /* USER_DEFINES_H_ */
```

Listing A.64: main.c

```c
#include "defines.h"
#include "BMS.h"

int main(void)
{
   //Main loop.
   while(TRUE)
   {
      BMS();
   }
}
```

Listing A.65: syscalls.c

```
/*****************************************************************************/
    /*****
 * @file     stdio.c
 * @brief    Implementation of newlib syscall
 ******************************************************************************
     */

#include <stdio.h>
#include <stdarg.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "stm32f10x_usart.h"

#undef errno
extern int errno;
extern int  _end;

__attribute__ ((used))
caddr_t _sbrk ( int incr )
{
  static unsigned char *heap = NULL;
  unsigned char *prev_heap;

  if (heap == NULL) {
    heap = (unsigned char *)&_end;
  }
  prev_heap = heap;

  heap += incr;

  return (caddr_t) prev_heap;
}

__attribute__ ((used))
int link(char *old, char *new) {
return -1;
}

__attribute__ ((used))
int _close(int file)
{
  return -1;
}

__attribute__ ((used))
int _fstat(int file, struct stat *st)
{
  st->st_mode = S_IFCHR;
  return 0;
}

__attribute__ ((used))
int _isatty(int file)
{
  return 1;
}

__attribute__ ((used))
int _lseek(int file, int ptr, int dir)
{
```

```
59    return 0;
60  }
61  __attribute__ ((used))
62  int _read(int file, char *ptr, int len)
63  {
64    return 0;
65  }
66  __attribute__ ((used))
67  int _write(int file, char *ptr, int len)
68  {
69     int i;
70
71     for ( i = 0; i < len; i++)
72     {
73        USART_SendData(USART1, ptr[i]);
74
75        //Wait for message to be sent.
76        while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
77     }
78
79     return len;
80  }
81
82  __attribute__ ((used))
83  void abort(void)
84  {
85    /* Abort called */
86    while(1);
87  }
88
89  /* ---------------------------------- End Of File
       ------------------------------ */
```

# Appendix B

# GUI Code

## B.1 Qt Project file (.pro)

.

Listing B.1: BioimpedanceMeasurementSystem.pro

```
1  QT       += core gui
2
3  greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport
4
5  TARGET = BioimpedanceMeasurementSystem
6  TEMPLATE = app
7
8  INCLUDEPATH += $$PWD/QCustomPlot
9
10 SOURCES += main.cpp\
11         mainwindow.cpp \
12     bluetoothdevicelist.cpp \
13     bluetooth.cpp \
14     measurementplots.cpp \
15     QCustomPlot/qcustomplot.cpp \
16     receive.cpp \
17     send.cpp
18
19 HEADERS  += mainwindow.h \
20     bluetoothdevicelist.h \
21     bluetooth.h \
22     measurementplots.h \
23     QCustomPlot/qcustomplot.h \
24     menuEnums.h \
25     messages.h \
26     receive.h \
27     send.h
28
29 FORMS    += mainwindow.ui \
30     measurementplots.ui
31
32 #Different platform sources, Qt modules and other files.
33 android {
34     QT += androidextras
35     SOURCES += bluetooth_android.cpp
```

```
36      OTHER_FILES += android/AndroidManifest.xml\
37      android/src/org/qtproject/qt5/android/bindings/MyActivity.java
38
39  } else {
40      QT += serialport
41      SOURCES += bluetooth_desktop.cpp
42  }
43
44  CONFIG += mobility c++11
45  MOBILITY =
46
47  #Path to Android files.
48  ANDROID_PACKAGE_SOURCE_DIR = $$PWD/android
49
50  RESOURCES += \
51      resources.qrc
```

## B.2 Code: C++

.

Listing B.2: main.cpp

```
1  #include "mainwindow.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MainWindow w;
8      w.showMaximized();
9      return a.exec();
10 }
```

Listing B.3: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSpinBox>
#include <QAction>
#include <QLabel>
#include "measurementplots.h"
#include "bluetoothdevicelist.h"
#include "bluetooth.h"
#include "send.h"
#include "receive.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    void setupMenuAndToolBar();

    //Menu items.
    QMenu *discover;
    QMenu *togglePlotView;
    QMenu *realTimePlot;
    QMenu *resistanceReactance;
    QMenu *conductanceSuceptance;
    QMenu *modulusPhase;
    QMenu *spectroscopyPlot;
    QMenu *fourFSMeasurement;
    QMenu *normalMeasurement;
    QMenu *frequency;
    QMenu *calibration;
    QMenu *about;

    //Menu and toolbar actions.
    QActionGroup *discoverToggle;
    QAction *discoverAction;
    QAction *togglePlotViewAction;

    QAction *realTimePlotAction;

    QActionGroup *measurementParameter;
    QAction *resistanceReactanceAction;
    QAction *conductanceSuceptanceAction;
    QAction *modulusPhaseAction;

    QAction *spectroscopyPlotAction;

    QActionGroup *measurementTechnique;
    QAction *fourFSMeasurementAction;
    QAction *normalMeasurementAction;
```

```
61
62      QAction *frequencyAction;
63      QAction *calibrationAction;
64      QAction *aboutAction;
65
66      QSpinBox *frequencyBox;
67      QComboBox *systemModeBox;
68
69      //Status bar items.
70      QLabel *systemStatus;
71      QLabel *bluetoothStatus;
72      QLabel *calibrationStatus;
73      QLabel *resistanceStatus;
74      QLabel *reactanceStatus;
75      QLabel *conductanceStatus;
76      QLabel *susceptanceStatus;
77      QLabel *impedanceStatus;
78      QLabel *admittanceStatus;
79      QLabel *phaseStatus;
80
81      //Other
82      QStackedWidget *stackedWidget;
83      MeasurementPlots *measurementPlots;
84      BluetoothDeviceList *bluetoothDeviceList;
85      Bluetooth *bluetooth;
86      Send *send;
87      Receive *receive;
88      QPushButton *confirmButton;
89      QLabel *eggDemo;
90      unsigned int counter;
91      QTimer *heartbeat;
92
93  private slots:
94      void onDiscover();
95      void onTogglePlotView();
96      void onRealtimePlot(bool status);
97      void onResistanceReactance();
98      void onConductanceSusceptance();
99      void onModulusPhase();
100     void onSpectroscopyPlot(bool status);
101     void onFourFSMeasurement();
102     void onNormalMeasurement();
103     void onFrequencyValueChanged(int value);
104     void onSystemModeChanged(QString text);
105     void onCalibration();
106     void onAbout();
107
108     void showStatusBarText(QString text, int timeout = 3000);
109
110     void onBluetoothSearchDone();
111     void onDiscoveryStarted();
112     void onBluetoothConnected();
113     void onBluetoothDisconnected();
114     void onBluetoothAdapterNotFound();
115     void onBluetoothConnect();
116     void onHeartBeatReceived();
117     void onHeartBeatTimeout();
118     void onCalibrationStatusReceived(QString state);
119     void onConfirm();
120
121     void onResistanceReactanceReceived(QString resistance, QString reactance
            );
```

```
122     void onSpectroscopyDataReceived(QString frequency, QString resistance,
            QString reactance);
123     void onEggState(QString state);
124
125 signals:
126     void clearDeviceList();
127     void sendMeasurementSettingsMessage(MessageMeasurementSettings);
128     void modulusPhaseUpdated(QString, QString);
129     void spectroscopyDataUpdated(QString frequency, QString impedance,
            QString phase);
130 };
131
132 #endif // MAINWINDOW_H
```

Listing B.4: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
#include <QSpinBox>
#include <QMessageBox>
#include <QLabel>
#include <QPushButton>
#include <QComboBox>
#include <QPalette>
#include "bluetoothdevicelist.h"
#include "measurementplots.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    setWindowTitle("Bioimpedance Measurement System");

    setWindowIcon(QIcon(":/icons/resources/icons/appIcon.png"));

    counter = 0;

    stackedWidget = new QStackedWidget();

    //Create graph widget.
    measurementPlots = new MeasurementPlots();

    //Create bluetooth device list widget.
    bluetoothDeviceList = new BluetoothDeviceList();

    //Create Egg Demo widget.
    eggDemo = new QLabel("<font color='red'>BLACK = NEEDLE NOT IN EGG<br><br
        >WHITE = NEEDLE IN EGG WHITE<br><br>LIGHT YELLOW = NEEDLE BETWEEN
        EGG WHITE AND YOLK<br><br>YELLOW = EGG YOLK</font>");
    QFont eggDemoFont;
    eggDemoFont.setPointSize(30);
    eggDemo->setFont(eggDemoFont);
    QPalette eggDemoPalette;
    eggDemoPalette.setColor(QPalette::Window, Qt::black);
    eggDemo->setAutoFillBackground(true);
    eggDemo->setPalette(eggDemoPalette);
    eggDemo->setAlignment(Qt::AlignCenter);

    //Add widgets to a stacked widget.
    stackedWidget->addWidget(bluetoothDeviceList);
    stackedWidget->addWidget(measurementPlots);
    stackedWidget->addWidget(eggDemo);

    stackedWidget->setCurrentIndex(0);

    //Set graph widget as central widget.
    setCentralWidget(stackedWidget);

    //Initialize menu and tool bar.
    setupMenuAndToolBar();

    bluetooth = new Bluetooth();
    send = new Send();
```

```
59      receive = new Receive();
60
61      heartbeat = new QTimer();
62      connect(heartbeat, SIGNAL(timeout()), this, SLOT(onHeartBeatTimeout()));
63      heartbeat->start(5000);
64
65      connect(bluetooth, SIGNAL(bluetoothDeviceFound(QString)),
            bluetoothDeviceList, SLOT(addDevice(QString)));
66      connect(bluetooth, SIGNAL(bluetoothDiscoveryStarted()), this, SLOT(
            onDiscoveryStarted()));
67      connect(bluetooth, SIGNAL(bluetoothSearchDone()), this, SLOT(
            onBluetoothSearchDone()));
68      connect(bluetooth, SIGNAL(clearDiscoveryList()), bluetoothDeviceList,
            SLOT(clearDevices()));
69      connect(bluetooth, SIGNAL(connected()), this, SLOT(onBluetoothConnected
            ()));
70      connect(bluetooth, SIGNAL(couldNotConnect()), this, SLOT(
            onBluetoothDisconnected()));
71      connect(bluetooth, SIGNAL(bluetoothAdapterNotFound()), this, SLOT(
            onBluetoothAdapterNotFound()));
72      connect(bluetoothDeviceList, SIGNAL(bluetoothConnect()), this, SLOT(
            onBluetoothConnect()));
73      connect(bluetoothDeviceList, SIGNAL(setSelectedDevice(QString)),
            bluetooth, SLOT(setSelectedDevice(QString)));
74
75      connect(this, SIGNAL(clearDeviceList()), bluetoothDeviceList, SLOT(
            clearDevices()));
76      connect(bluetooth, SIGNAL(serialDataReady(QString)), receive, SLOT(
            addData(QString)));
77      connect(bluetooth, SIGNAL(bluetoothDataReceived(QString)), receive, SLOT
            (addData(QString)));
78
79      connect(receive, SIGNAL(measurementFrequencyChanged(QString)),
            measurementPlots, SLOT(changeRealtimePlotTitle(QString)));
80
81      //Measurement Settings.
82      connect(this , SIGNAL(sendMeasurementSettingsMessage(
            MessageMeasurementSettings)), send, SLOT(measurementSettings(
            MessageMeasurementSettings)));
83
84      //Packet.
85      //connect(receive, SIGNAL(frequencyAmplitudePhaseDataReceived(QString,
            QString,QString)), measurementPlots, SLOT(staticPlotDataSlot(QString
            ,QString,QString)));
86
87      connect(receive, SIGNAL(resistanceReactance(QString,QString)), this,
            SLOT(onResistanceReactanceReceived(QString,QString)));
88      connect(this, SIGNAL(modulusPhaseUpdated(QString, QString)),
            measurementPlots, SLOT(realtimePlotDataSlot(QString,QString)));
89
90      connect(receive, SIGNAL(spectropscopyDataUpdate(QString,QString,QString)
            ), this, SLOT(onSpectroscopyDataReceived(QString,QString,QString)));
91      connect(this, SIGNAL(spectroscopyDataUpdated(QString,QString,QString)),
            measurementPlots, SLOT(staticPlotDataSlot(QString,QString,QString)))
            ;
92
93      //connect(receive, SIGNAL(amplitudeAndPhaseDataReceived(QString,QString)
            ), topMenu, SLOT(on_amplitudePhaseReceived(QString,QString)));
94
95      //Send.
96      connect(send, SIGNAL(sendMeasurementSettings(QByteArray)), bluetooth,
            SLOT(write(QByteArray)));
```

```
97
98      connect(receive, SIGNAL(eggState(QString)), this, SLOT(onEggState(
            QString)));
99
100     connect(receive, SIGNAL(heartbeat()), this, SLOT(onHeartBeatReceived()))
            ;
101
102     connect(receive, SIGNAL(calibrationStatus(QString)), this, SLOT(
            onCalibrationStatusReceived(QString)));
103 }
104
105 MainWindow::~MainWindow()
106 {
107     delete ui;
108 }
109
110 //Sets up main windows menu and tool bars.
111 void MainWindow::setupMenuAndToolBar()
112 {
113     qDebug() << "Setting up menu, status and tool bar";
114
115     //Hide menu bar.
116     menuBar()->setVisible(false);
117
118     //Set tool bar static embedded within application window.
119     ui->mainToolBar->setMovable(false);
120
121     //Make sure toolbar items fills the toolbar in horizontal direction.
122     ui->mainToolBar->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::
            Preferred);
123
124     //Create Discover menu.
125     discover = menuBar()->addMenu(tr("&Discover"));
126
127     //Create Discover action. Set discover icon, priority and shortcut.
            Connect signal to slot and add to menu and tool bar.
128     discoverAction = discover->addAction(tr("&Discover"));
129     discoverAction->setIcon(QIcon(":/icons/resources/icons/discover.png"));
130     discoverAction->setPriority(QAction::HighPriority);
131     discoverAction->setShortcut(QKeySequence::New);
132     ui->mainToolBar->addAction(discoverAction);
133     discover->addAction(discoverAction);
134     connect(discoverAction, SIGNAL(triggered()), this, SLOT(onDiscover()));
135
136     ui->mainToolBar->addSeparator();
137
138     //Create TogglePlotView menu.
139     togglePlotView = menuBar()->addMenu(tr("&Toggle Plot View"));
140
141     //Create TogglePlotView action. Set discover icon, priority and shortcut
            . Connect signal to slot and add to menu and tool bar.
142     togglePlotViewAction = togglePlotView->addAction(tr("&Toggle Plot View")
            );
143     togglePlotViewAction->setIcon(QIcon(":/icons/resources/icons/toggleChart
            .png"));
144     togglePlotViewAction->setPriority(QAction::HighPriority);
145     togglePlotViewAction->setShortcut(QKeySequence::New);
146     ui->mainToolBar->addAction(togglePlotViewAction);
147     togglePlotView->addAction(togglePlotViewAction);
148     connect(togglePlotViewAction, SIGNAL(triggered()), this, SLOT(
            onTogglePlotView()));
149
```

```
150     discoverToggle = new QActionGroup(this);
151     discoverToggle->addAction(discoverAction);
152     discoverToggle->addAction(togglePlotViewAction);
153
154     ui->mainToolBar->addSeparator();
155
156     //Create SpectroscopyPlot menu.
157     spectroscopyPlot = menuBar()->addMenu(tr("&Spectroscopy Plot"));
158
159     //Create SpectroscopyPlot action. Set discover icon, priority and
            shortcut. Connect signal to slot and add to menu and tool bar.
160     spectroscopyPlotAction = spectroscopyPlot->addAction(tr("&Spectroscopy
            Plot"));
161     spectroscopyPlotAction->setIcon(QIcon(":/icons/resources/icons/fChart.
            png"));
162     spectroscopyPlotAction->setPriority(QAction::HighPriority);
163     spectroscopyPlotAction->setShortcut(QKeySequence::New);
164     ui->mainToolBar->addAction(spectroscopyPlotAction);
165     spectroscopyPlot->addAction(spectroscopyPlotAction);
166     spectroscopyPlotAction->setCheckable(true);
167     spectroscopyPlotAction->setChecked(false);
168     connect(spectroscopyPlotAction, SIGNAL(toggled(bool)), this, SLOT(
            onSpectroscopyPlot(bool)));
169
170     ui->mainToolBar->addSeparator();
171
172     //Create RealtimePlot menu.
173     realTimePlot = menuBar()->addMenu(tr("&Realtime Plot"));
174
175     //Create RealtimePlot action. Set discover icon, priority and shortcut.
            Connect signal to slot and add to menu and tool bar.
176     realTimePlotAction = realTimePlot->addAction(tr("&Realtime Plot"));
177     realTimePlotAction->setIcon(QIcon(":/icons/resources/icons/tChart.png"))
            ;
178     realTimePlotAction->setPriority(QAction::HighPriority);
179     realTimePlotAction->setShortcut(QKeySequence::New);
180     ui->mainToolBar->addAction(realTimePlotAction);
181     realTimePlot->addAction(realTimePlotAction);
182     realTimePlotAction->setCheckable(true);
183     realTimePlotAction->setChecked(false);
184     connect(realTimePlotAction, SIGNAL(toggled(bool)), this, SLOT(
            onRealtimePlot(bool)));
185
186
187     //Create ResistanceReactance menu.
188     resistanceReactance = menuBar()->addMenu(tr("&Resistance && Reactance"))
            ;
189
190     //Create ResistanceReactance action. Set discover icon, priority and
            shortcut. Connect signal to slot and add to menu and tool bar.
191     resistanceReactanceAction = resistanceReactance->addAction(tr("&
            Resistance && Reactance"));
192     resistanceReactanceAction->setIcon(QIcon(":/icons/resources/icons/
            resistanceReactance.png"));
193     resistanceReactanceAction->setPriority(QAction::HighPriority);
194     resistanceReactanceAction->setShortcut(QKeySequence::New);
195     //ui->mainToolBar->addAction(resistanceReactanceAction);
196     resistanceReactance->addAction(resistanceReactanceAction);
197     resistanceReactanceAction->setCheckable(true);
198     connect(resistanceReactanceAction, SIGNAL(triggered()), this, SLOT(
            onResistanceReactance()));
199
```

```
200
201     //Create ConductanceSusceptance menu.
202     conductanceSuceptance = menuBar()->addMenu(tr("&Conductance &&
            Susceptance"));
203
204     //Create ConductanceSusceptance action. Set discover icon, priority and
            shortcut. Connect signal to slot and add to menu and tool bar.
205     conductanceSuceptanceAction = conductanceSuceptance->addAction(tr("&
            Conductance && Susceptance"));
206     conductanceSuceptanceAction->setIcon(QIcon(":/icons/resources/icons/
            conductanceSusceptance.png"));
207     conductanceSuceptanceAction->setPriority(QAction::HighPriority);
208     conductanceSuceptanceAction->setShortcut(QKeySequence::New);
209     //ui->mainToolBar->addAction(conductanceSuceptanceAction);
210     conductanceSuceptance->addAction(conductanceSuceptanceAction);
211     conductanceSuceptanceAction->setCheckable(true);
212     connect(conductanceSuceptanceAction, SIGNAL(triggered()), this, SLOT(
            onConductanceSusceptance()));
213
214
215     //Create ModulusPhase menu.
216     modulusPhase = menuBar()->addMenu(tr("&Modulus && Phase"));
217
218     //Create ModulusPhase action. Set discover icon, priority and shortcut.
            Connect signal to slot and add to menu and tool bar.
219     modulusPhaseAction = modulusPhase->addAction(tr("&Modulus && Phase"));
220     modulusPhaseAction->setIcon(QIcon(":/icons/resources/icons/modulusPhase.
            png"));
221     modulusPhaseAction->setPriority(QAction::HighPriority);
222     modulusPhaseAction->setShortcut(QKeySequence::New);
223     //ui->mainToolBar->addAction(modulusPhaseAction);
224     modulusPhase->addAction(modulusPhaseAction);
225     modulusPhaseAction->setCheckable(true);
226     connect(modulusPhaseAction, SIGNAL(triggered()), this, SLOT(
            onModulusPhase()));
227
228     measurementParameter = new QActionGroup(this);
229     measurementParameter->addAction(resistanceReactanceAction);
230     measurementParameter->addAction(conductanceSuceptanceAction);
231     measurementParameter->addAction(modulusPhaseAction);
232
233     modulusPhaseAction->setChecked(true);
234
235     ui->mainToolBar->addSeparator();
236
237     //Create Frequency toolbar item.
238     frequencyBox = new QSpinBox();
239     frequencyBox->setRange(100, 140000);
240     frequencyBox->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::
            Expanding);
241     QFont freNumfont;
242     freNumfont.setPointSize(30);
243     frequencyBox->setFont(freNumfont);
244     ui->mainToolBar->addWidget(frequencyBox);
245     connect(frequencyBox, SIGNAL(valueChanged(int)), this, SLOT(
            onFrequencyValueChanged(int)));
246
247     ui->mainToolBar->addSeparator();
248
249     //Create SystemMode toolbar item.
250     systemModeBox = new QComboBox();
251     systemModeBox->addItem("Idle");
```

```
252     systemModeBox->addItem("Measurement Mode");
253     systemModeBox->addItem("Egg Demo Mode");
254     systemModeBox->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::
            Expanding);
255     QFont sysModefont;
256     sysModefont.setPointSize(20);
257     systemModeBox->setFont(sysModefont);
258     ui->mainToolBar->addWidget(systemModeBox);
259     connect(systemModeBox, SIGNAL(currentIndexChanged(QString)), this, SLOT(
            onSystemModeChanged(QString)));
260
261
262     //Create 4fsMeasurement menu.
263     fourFSMeasurement = menuBar()->addMenu(tr("&4fs Measurement"));
264
265     //Create 4fsMeasurement action. Set discover icon, priority and shortcut
            . Connect signal to slot and add to menu and tool bar.
266     fourFSMeasurementAction = fourFSMeasurement->addAction(tr("&4fs
            Measurement"));
267     fourFSMeasurementAction->setIcon(QIcon(":/icons/resources/icons/4fs.png"
            ));
268     fourFSMeasurementAction->setPriority(QAction::HighPriority);
269     fourFSMeasurementAction->setShortcut(QKeySequence::New);
270     ui->mainToolBar->addAction(fourFSMeasurementAction);
271     fourFSMeasurement->addAction(fourFSMeasurementAction);
272     fourFSMeasurementAction->setCheckable(true);
273     connect(fourFSMeasurementAction, SIGNAL(triggered()), this, SLOT(
            onFourFSMeasurement()));
274
275
276     //Create NormalMeasurement menu.
277     normalMeasurement = menuBar()->addMenu(tr("&Normal Measurement"));
278
279     //Create NormalMeasurement action. Set discover icon, priority and
            shortcut. Connect signal to slot and add to menu and tool bar.
280     normalMeasurementAction = normalMeasurement->addAction(tr("&Normal
            Measurement"));
281     normalMeasurementAction->setIcon(QIcon(":/icons/resources/icons/normal.
            png"));
282     normalMeasurementAction->setPriority(QAction::HighPriority);
283     normalMeasurementAction->setShortcut(QKeySequence::New);
284     ui->mainToolBar->addAction(normalMeasurementAction);
285     normalMeasurement->addAction(normalMeasurementAction);
286     normalMeasurementAction->setCheckable(true);
287     connect(normalMeasurementAction, SIGNAL(triggered()), this, SLOT(
            onNormalMeasurement()));
288
289     measurementTechnique = new QActionGroup(this);
290     measurementTechnique->addAction(fourFSMeasurementAction);
291     measurementTechnique->addAction(normalMeasurementAction);
292
293     fourFSMeasurementAction->setChecked(true);
294
295     ui->mainToolBar->addSeparator();
296
297     //Create Calibrate menu.
298     calibration = menuBar()->addMenu(tr("&Calibrate"));
299
300     //Create Calibrate action. Set discover icon, priority and shortcut.
            Connect signal to slot and add to menu and tool bar.
301     calibrationAction = calibration->addAction(tr("&Calibrate"));
```

```
302     calibrationAction->setIcon(QIcon(":/icons/resources/icons/calibration.
            png"));
303     calibrationAction->setPriority(QAction::HighPriority);
304     calibrationAction->setShortcut(QKeySequence::New);
305     ui->mainToolBar->addAction(calibrationAction);
306     calibration->addAction(calibrationAction);
307     calibrationAction->setCheckable(true);
308     calibrationAction->setChecked(false);
309     connect(calibrationAction, SIGNAL(triggered()), this, SLOT(onCalibration
            ()));
310
311     //Create About menu.
312     about = menuBar()->addMenu(tr("&About"));
313
314     //Create About action. Set discover icon, priority and shortcut. Connect
             signal to slot and add to menu and tool bar.
315     aboutAction = about->addAction(tr("&About"));
316     aboutAction->setIcon(QIcon(":/icons/resources/icons/about.png"));
317     aboutAction->setPriority(QAction::HighPriority);
318     aboutAction->setShortcut(QKeySequence::New);
319     connect(aboutAction, SIGNAL(triggered()), this, SLOT(onAbout()));
320     ui->mainToolBar->addAction(aboutAction);
321     about->addAction(aboutAction);
322
323     //Set Tool Bar maximum icon size.
324     ui->mainToolBar->setIconSize(QSize(128,128));
325
326     //Configure Status Bar.
327
328     //Add confirm button.
329     confirmButton = new QPushButton();
330     confirmButton->setText("Confirm");
331     confirmButton->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::
            Expanding);
332     confirmButton->setFixedWidth(250);
333     ui->statusBar->addPermanentWidget(confirmButton);
334     confirmButton->setEnabled(false);
335     connect(confirmButton, SIGNAL(pressed()), this, SLOT(onConfirm()));
336
337     //Add system status.
338     systemStatus = new QLabel();
339     ui->statusBar->addPermanentWidget(systemStatus);
340     systemStatus->setPixmap(QPixmap(":/icons/resources/icons/
            systemStatusError.png"));
341
342     //Add bluetooth status.
343     bluetoothStatus = new QLabel();
344     ui->statusBar->addPermanentWidget(bluetoothStatus);
345     bluetoothStatus->setPixmap(QPixmap(":/icons/resources/icons/
            bluetoothDisconnected.png"));
346
347     //Add calibration status.
348     calibrationStatus = new QLabel();
349     ui->statusBar->addPermanentWidget(calibrationStatus);
350     calibrationStatus->setPixmap(QPixmap(":/icons/resources/icons/
            calibrationNotDone.png"));
351
352     ui->statusBar->addPermanentWidget(new QLabel("<b>R: </b>"));
353     resistanceStatus = new QLabel();
354     resistanceStatus->setFixedWidth(75);
355     ui->statusBar->addPermanentWidget(resistanceStatus);
356
```

```
357     ui->statusBar->addPermanentWidget(new QLabel("<b>X: </b>"));
358     reactanceStatus = new QLabel();
359     reactanceStatus->setFixedWidth(75);
360     ui->statusBar->addPermanentWidget(reactanceStatus);
361
362     ui->statusBar->addPermanentWidget(new QLabel("<b>G [uS]: </b>"));
363     conductanceStatus = new QLabel();
364     conductanceStatus->setFixedWidth(75);
365     ui->statusBar->addPermanentWidget(conductanceStatus);
366
367     ui->statusBar->addPermanentWidget(new QLabel("<b>B [uS]: </b>"));
368     susceptanceStatus = new QLabel();
369     susceptanceStatus->setFixedWidth(75);
370     ui->statusBar->addPermanentWidget(susceptanceStatus);
371
372     ui->statusBar->addPermanentWidget(new QLabel("<b>Z: </b>"));
373     impedanceStatus = new QLabel();
374     impedanceStatus->setFixedWidth(75);
375     ui->statusBar->addPermanentWidget(impedanceStatus);
376
377     ui->statusBar->addPermanentWidget(new QLabel("<b>Y: [uS]</b>"));
378     admittanceStatus = new QLabel();
379     admittanceStatus->setFixedWidth(75);
380     ui->statusBar->addPermanentWidget(admittanceStatus);
381
382     QString temp;
383     temp.append("<b>");
384     temp.append(QChar(0x03B8));
385     temp.append(": ");
386     temp.append("</b>");
387     ui->statusBar->addPermanentWidget(new QLabel(temp));
388     phaseStatus = new QLabel();
389     phaseStatus->setFixedWidth(75);
390     ui->statusBar->addPermanentWidget(phaseStatus);
391
392     resistanceStatus->setText("0");
393     reactanceStatus->setText("0");
394     conductanceStatus->setText("0");
395     susceptanceStatus->setText("0");
396     impedanceStatus->setText("0");
397     admittanceStatus->setText("0");
398     phaseStatus->setText("0");
399 }
400
401 void MainWindow::onDiscover()
402 {
403     qDebug() << "Discover";
404
405     stackedWidget->setCurrentIndex(0);
406     bluetoothDeviceList->clear();
407     bluetooth->connect();
408     bluetooth->discover();
409 }
410
411 void MainWindow::onTogglePlotView()
412 {
413     qDebug() << "Toggle Plot View!";
414
415     if(stackedWidget->currentIndex() == 1)
416     {
417         measurementPlots->onTogglePlotView();
418     }
```

```
419     else
420     {
421         stackedWidget->setCurrentIndex(1);
422     }
423 }
424
425 void MainWindow::onRealtimePlot(bool status)
426 {
427     qDebug() << "Real Time Plot: " << status;
428     showStatusBarText("Real Time Plot " + ((status) ? QString("Enabled") :
        QString("Disabled")) );
429 }
430
431 void MainWindow::onResistanceReactance()
432 {
433     qDebug() << "Resistance & Reactance!";
434     showStatusBarText("Resistance & Reactance Selected");
435 }
436
437 void MainWindow::onConductanceSusceptance()
438 {
439     qDebug() << "Conductance & Susceptance!";
440     showStatusBarText("Conductance & Susceptance Selected");
441 }
442
443 void MainWindow::onModulusPhase()
444 {
445     qDebug() << "Modulus & Phase!";
446     showStatusBarText("Modulus & Phase Selected");
447 }
448
449 void MainWindow::onSpectroscopyPlot(bool status)
450 {
451     qDebug() << "Spectroscopy Plot: " << status;
452     showStatusBarText("Spectroscopy Plot " + ((status) ? QString("Enabled")
        : QString("Disabled")) );
453 }
454
455 void MainWindow::onFourFSMeasurement()
456 {
457     qDebug() << "4fs Measurement!";
458     showStatusBarText("4fs Measurement Selected");
459 }
460
461 void MainWindow::onNormalMeasurement()
462 {
463     qDebug() << "Normal Measurement!";
464     showStatusBarText("Normal Measurement Selected");
465
466 }
467
468 void MainWindow::onFrequencyValueChanged(int value)
469 {
470     qDebug() << QString::number(value);
471 }
472
473 void MainWindow::onSystemModeChanged(QString text)
474 {
475     qDebug() << "System mode: " << text;
476
477     if(text == "Measurement Mode")
478     {
```

```
479         stackedWidget->setCurrentIndex(1);
480     }
481     else if(text == "Egg Demo Mode")
482     {
483         stackedWidget->setCurrentIndex(2);
484     }
485 }
486
487 void MainWindow::onCalibration()
488 {
489     qDebug() << "Calibration";
490 }
491
492 //Shows the "about this application" widget.
493 void MainWindow::onAbout()
494 {
495     qDebug() << "about";
496     showStatusBarText("Showing About Window");
497
498     QMessageBox::about(this, tr("About: Bioimpedance Measurement System"),
        tr("Bioimpedance Measurement System 1.0\n\nThis application is
        written by Patrick Hisni Brataas.\nE-mail: patrick.brataas@live.com\
        n\nThe program is provided AS IS with NO WARRANTY OF ANY KIND,
        INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A
        PARTICULAR PURPOSE."));
499 }
500
501 void MainWindow::showStatusBarText(QString text, int timeout)
502 {
503     ui->statusBar->showMessage(text, timeout);
504 }
505
506 void MainWindow::onBluetoothSearchDone()
507 {
508     qDebug() << "Qt: Discovery Done";
509
510     //Notify the user that search is done.
511     if(bluetoothDeviceList->count() == 0)
512     {
513         showStatusBarText("Search done! No devices found. Please try again."
            );
514     }
515     else
516     {
517         showStatusBarText("Search done! " + QString::number(
            bluetoothDeviceList->count()) +  " devices found.");
518     }
519 }
520
521 void MainWindow::onDiscoveryStarted()
522 {
523     qDebug() << "Device discovery started";
524
525     showStatusBarText("Discovery started. Looking for devices...");
526 }
527
528 void MainWindow::onBluetoothConnected()
529 {
530     qDebug() << "Bluetooth device status: Connected";
531
532     showStatusBarText("Bluetooth device status: Connected");
```

```
533     bluetoothStatus->setPixmap(QPixmap(":/icons/resources/icons/
            bluetoothConnected.png"));
534     confirmButton->setEnabled(true);
535 }
536
537 void MainWindow::onBluetoothDisconnected()
538 {
539     qDebug() << "Bluetooth device status: Disconnected";
540
541     showStatusBarText("Bluetooth device status: Disconnected");
542     bluetoothStatus->setPixmap(QPixmap(":/icons/resources/icons/
            bluetoothDisconnected.png"));
543     confirmButton->setEnabled(false);
544 }
545
546 void MainWindow::onBluetoothAdapterNotFound()
547 {
548     qDebug() << "No bluetooth hardware support found";
549
550     showStatusBarText("Bluetooth hardware not found on this device.");
551 }
552
553 void MainWindow::onBluetoothConnect()
554 {
555     qDebug() << "Connecting to selected device";
556
557     bluetooth->deviceSelected();
558 }
559
560 void MainWindow::onConfirm()
561 {
562     MessageMeasurementSettings msg = {0, 0, 0, 0, 0, 0, 0, 0};
563
564     //Enable or disable real time plot.
565     if(realTimePlotAction->isChecked())
566     {
567         msg.realtimePlotEnable = ENABLE;
568     }
569     else
570     {
571         msg.realtimePlotEnable = DISABLE;
572     }
573
574     //Enable or disable spectroscopy plot.
575     if(spectroscopyPlotAction->isChecked())
576     {
577         msg.spectroscopyPlotEnable = ENABLE;
578     }
579     else
580     {
581         msg.spectroscopyPlotEnable = DISABLE;
582     }
583
584     //Real time plot mode. (RX, GB, ZPHI)
585 //    if(resistanceReactanceAction->isChecked())
586 //    {
587 //        msg.realtimePlotParameters = RESISTANCE_AND_REACTANCE;
588 //    }
589 //    else if(conductanceSuceptanceAction->isChecked())
590 //    {
591 //        msg.realtimePlotParameters = CONDUCTANCE_AND_SUSEPTANCE;
592 //    }
```

```
593 //    else if(modulusPhaseAction->isChecked())
594 //    {
595 //        msg.realtimePlotParameters = MODULUS_AND_PHASE;
596 //    }
597
598     //Frequency.
599     msg.frequency = (unsigned int) frequencyBox->value();
600
601     //4fs or Normal.
602     if(fourFSMeasurementAction->isChecked())
603     {
604         msg.technique = TECHNIQUE_4FS;
605     }
606     else if(normalMeasurementAction->isChecked())
607     {
608         msg.technique = TECHNIQUE_NORMAL;
609     }
610
611     if(calibrationAction->isChecked())
612     {
613         msg.calibratePhase = ENABLE;
614     }
615
616     qDebug() << QString::number(systemModeBox->currentIndex());
617
618     msg.systemMode = systemModeBox->currentIndex() + 1;
619
620     emit sendMeasurementSettingsMessage(msg);
621 }
622
623 void MainWindow::onResistanceReactanceReceived(QString resistance, QString
        reactance)
624 {
625     unsigned int _resistance;
626     int _reactance;
627     double dividend;
628     double conductance;
629     double susceptance;
630     int impedance;
631     double admittance;
632     double phase;
633
634     //Convert from string to numbers.
635     _resistance = resistance.toUInt();
636     _reactance = reactance.toInt();
637
638     //Calculate dividend once.
639     dividend = (pow((double)abs((int)_resistance), 2) + pow((double)abs(
            _reactance), 2));
640
641     //Calculate impedance modulus.
642     impedance = (int) sqrt(dividend);
643
644     //Calculate phase in degrees.
645     phase = atan2((double)_reactance, (double)_resistance) * (180 / M_PI);
646
647     counter++;
648
649     if( (counter % 15) == 0)
650     {
651         //Calculate conductance and susceptance in uS.
```

```
652         conductance = (double)((double)((double)_resistance * 1000000) /
                dividend);
653         susceptance = (double)((double)(_reactance * 1000000) / dividend) *
                -1;
654
655         //Calculate admittance modulus in uS.
656         admittance = (double)(1 / ((double)impedance / 1000000));
657
658         //Set resistance and reactance.
659         resistanceStatus->setText(resistance);
660         reactanceStatus->setText(reactance);
661
662         //Set conductance and susceptance.
663         conductanceStatus->setText(QString::number(conductance));
664         susceptanceStatus->setText(QString::number(susceptance));
665
666         //Set impedance and admittance modulus.
667         impedanceStatus->setText(QString::number(impedance));
668         admittanceStatus->setText(QString::number(admittance));
669
670         //Set phase.
671         phaseStatus->setText(QString::number(phase));
672     }
673
674     //Update plot.
675     emit modulusPhaseUpdated(QString::number(impedance), QString::number(
            phase));
676 }
677
678 void MainWindow::onSpectroscopyDataReceived(QString frequency, QString
        resistance, QString reactance)
679 {
680     unsigned int _resistance;
681     int _reactance;
682     double dividend;
683     int impedance;
684     double phase;
685
686     //Convert from string to numbers.
687     _resistance = resistance.toUInt();
688     _reactance = reactance.toInt();
689
690     //Calculate dividend once.
691     dividend = (pow((double)abs((int)_resistance), 2) + pow((double)abs(
            _reactance), 2));
692
693     //Calculate impedance modulus.
694     impedance = (int) sqrt(dividend);
695
696     //qDebug() << QString::number(impedance);
697
698     //Calculate phase in degrees.
699     phase = atan2((double)_reactance, (double)_resistance) * (180 / M_PI);
700
701     emit spectroscopyDataUpdated(frequency, QString::number(impedance),
            QString::number(phase));
702 }
703
704 void MainWindow::onEggState(QString state)
705 {
706     QPalette eggDemoPalette;
707
```

```
708     switch (state.toUInt())
709     {
710     case 0:
711         //Needle not in egg yet.
712 
713         eggDemoPalette.setColor(QPalette::Window, Qt::black);
714         eggDemo->setPalette(eggDemoPalette);
715         break;
716 
717     case 1:
718         //Egg white.
719         eggDemoPalette.setColor(QPalette::Window, Qt::white);
720         eggDemo->setPalette(eggDemoPalette);
721         break;
722 
723     case 2:
724         //Egg yolk.
725         eggDemoPalette.setColor(QPalette::Window, Qt::yellow);
726         eggDemo->setPalette(eggDemoPalette);
727         break;
728 
729     case 3:
730         //In between.
731         eggDemoPalette.setColor(QPalette::Window, QColor(0xFF, 0x66, 0x00));
732         eggDemo->setPalette(eggDemoPalette);
733         break;
734     default:
735         break;
736     }
737 }
738 
739 void MainWindow::onHeartBeatReceived()
740 {
741     //Restart the heartbeat timer.
742     heartbeat->start(10000);
743 
744     //Change icon to system running.
745     systemStatus->setPixmap(QPixmap(":/icons/resources/icons/systemStatusOK.
        png"));
746 }
747 
748 void MainWindow::onHeartBeatTimeout()
749 {
750     //Change icon to system not running.
751     systemStatus->setPixmap(QPixmap(":/icons/resources/icons/
        systemStatusError.png"));
752 }
753 
754 void MainWindow::onCalibrationStatusReceived(QString state)
755 {
756     //Update calibration icon.
757     if(state.toUInt() == 0)
758     {
759         calibrationStatus->setPixmap(QPixmap(":/icons/resources/icons/
            calibrationNotDone.png"));
760     }
761     else if (state.toUInt() == 1)
762     {
763         calibrationStatus->setPixmap(QPixmap(":/icons/resources/icons/
            calibrationDone.png"));
764     }
765 }
```

Listing B.5: bluetooth.h

```
#ifndef BLUETOOTH_H
#define BLUETOOTH_H

#include <QWidget>
#include <QString>
#include <QListWidget>

struct QSerialPort;
struct QSerialPortInfo;

class Bluetooth : public QWidget
{
    Q_OBJECT
public:
    explicit Bluetooth(QWidget *parent = 0);
    void connect();
    void discover();
    void sendData(QString string);
    void deviceSelected();

    static Bluetooth *instance() { return singletonInstance; }

signals:
    //Java signals
    void bluetoothDeviceFound(QString device);
    void bluetoothDiscoveryStarted();
    void bluetoothSearchDone();
    void bluetoothAdapterNotFound();
    void bluetoothDataReceived(QString data);

    void bluetoothExceptionCreateRFCOMM();
    void bluetoothExceptionJoinThread();
    void bluetoothExceptionInputOutputStream();
    void bluetoothExceptionReadInputStream();
    void bluetoothExceptionWriteOutputStream();

    void serialDataReady(QString data);
    void clearDiscoveryList();

    void connected();
    void couldNotConnect();

public slots:
    void setSelectedDevice(QString device);
    void write(QString macAddress);
    void write(QByteArray byteArray);

private slots:
    void serialDataRead();
    void on_couldNotConnect();
    void on_connected();
    void on_portSettingsFailed();
    void on_openPort();

private:
    static Bluetooth *singletonInstance;
    QList<QSerialPortInfo> *ports;
    QString *bluetoothSelectedDevice;
    QSerialPort *serialPort;
    QThread *serialWorkerThread;
```

```
61
62 };
63
64 #endif // BLUETOOTH_H
```

Listing B.6: bluetooth.cpp

```
1 #include "bluetooth.h"
2 #include <QString>
3
4 Bluetooth *Bluetooth::singletonInstance = 0;
5
6 void Bluetooth::setSelectedDevice(QString device)
7 {
8     *bluetoothSelectedDevice = device;
9 }
```

Listing B.7: bluetooth_android.cpp

```
1  #include "bluetooth.h"
2  #include <QWidget>
3  #include <QtAndroidExtras/QAndroidJniObject>
4  #include <QString>
5  #include <QDebug>
6
7  /****************************************/
8  /** Android specific bluetooth methods. **/
9  /****************************************/
10
11 Bluetooth::Bluetooth(QWidget *parent) :
12     QWidget(parent)
13 {
14     //Initialize singleton.
15     singletonInstance = this;
16     bluetoothSelectedDevice = new QString();
17 }
18
19 void Bluetooth::sendData(QString string)
20 {
21 }
22
23 ///
24 /// \brief Bluetooth::connect
25 ///
26 void Bluetooth::connect()
27 {
28     //Connect
29     QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
           bindings/MyActivity", "BluetoothConnect");
30 }
31
32 ///
33 /// \brief Bluetooth::discover
34 ///
35 void Bluetooth::discover()
36 {
37     //Discover
38     QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
           bindings/MyActivity", "BluetoothStartDiscovery");
39 }
40
41 ///
42 /// \brief Bluetooth::write
43 /// \param string
44 ///
45 void Bluetooth::write(QString string)
46 {
47     //Convert QString to Java string.
48     QAndroidJniObject javaString = QAndroidJniObject::fromString(string);
49
50     //Send data
51     QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
           bindings/MyActivity", "BluetoothWrite", "(Ljava/lang/String;)V",
           javaString.object<jstring>());
52 }
53
54 ///
55 /// \brief Bluetooth::deviceSelected
56 /// \param macAddress
```

```
57  ///
58  void Bluetooth::deviceSelected()
59  {
60      //Parse item for MAC address.
61      QString macAddress = bluetoothSelectedDevice->mid( (
            bluetoothSelectedDevice->length() - 17) );
62      qDebug() << "MAC address: " + macAddress;
63
64      //Stop discovery if still discovering.
65      QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
            bindings/MyActivity", "BluetoothCancelDiscovery");
66
67      //Convert QString to Java string.
68      QAndroidJniObject javaString = QAndroidJniObject::fromString(macAddress)
            ;
69
70      //Send MAC address to start establishing a connection.
71      QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
            bindings/MyActivity", "BluetoothConnectToDevice", "(Ljava/lang/
            String;)V", javaString.object<jstring>());
72
73      //Emit signal that bluetooth is connected.
74      emit connected();
75  }
76
77  /********************/
78  /** Java signals. **/
79  /********************/
80
81  ///
82  /// \brief bluetoothDeviceFound
83  /// \param env
84  /// \param device
85  ///
86  static void bluetoothDeviceFound(JNIEnv * env, jclass /*clazz*/, jstring
        device)
87  {
88      QString temp = env->GetStringUTFChars(device, 0);
89      if(temp.isNull() || temp.isEmpty())
90      {
91          return;
92      }
93      Bluetooth::instance()->bluetoothDeviceFound(temp);
94      env->ReleaseStringUTFChars(device, temp.toUtf8());
95  }
96
97  ///
98  /// \brief bluetoothDiscoveryStarted
99  ///
100 static void bluetoothDiscoveryStarted(JNIEnv * /*env*/, jclass /*clazz*/)
101 {
102     Bluetooth::instance()->bluetoothDiscoveryStarted();
103 }
104
105 ///
106 /// \brief bluetoothSearchDone
107 ///
108 static void bluetoothSearchDone(JNIEnv * /*env*/, jclass /*clazz*/)
109 {
110     Bluetooth::instance()->bluetoothSearchDone();
111 }
112
```

```
113 ///
114 /// \brief bluetoothAdapterNotFound
115 ///
116 static void bluetoothAdapterNotFound(JNIEnv * /*env*/, jclass /*clazz*/)
117 {
118     Bluetooth::instance()->bluetoothAdapterNotFound();
119 }
120
121 ///
122 /// \brief bluetoothDataReceived
123 /// \param env
124 /// \param data
125 ///
126 static void bluetoothDataReceived(JNIEnv * env, jclass /*clazz*/, jstring
        data)
127 {
128     const char *temp = env->GetStringUTFChars(data, 0);
129     QString tempS(temp);
130     if(tempS.isNull() || tempS.isEmpty())
131     {
132         return;
133     }
134     Bluetooth::instance()->bluetoothDataReceived(tempS);
135     env->ReleaseStringUTFChars(data, temp);
136 }
137
138 ///
139 /// \brief bluetoothExceptionCreateRFCOMM
140 ///
141 static void bluetoothExceptionCreateRFCOMM(JNIEnv * /*env*/, jclass /*clazz
        */)
142 {
143     Bluetooth::instance()->bluetoothExceptionCreateRFCOMM();
144 }
145
146 ///
147 /// \brief bluetoothExceptionJoinThread
148 ///
149 static void bluetoothExceptionJoinThread(JNIEnv * /*env*/, jclass /*clazz*/)
150 {
151     Bluetooth::instance()->bluetoothExceptionJoinThread();
152 }
153
154 ///
155 /// \brief bluetoothExceptionInputOutputStream
156 ///
157 static void bluetoothExceptionInputOutputStream(JNIEnv * /*env*/, jclass /*
        clazz*/)
158 {
159     Bluetooth::instance()->bluetoothExceptionInputOutputStream();
160 }
161
162 ///
163 /// \brief bluetoothExceptionReadInputStream
164 ///
165 static void bluetoothExceptionReadInputStream(JNIEnv * /*env*/, jclass /*
        clazz*/)
166 {
167     Bluetooth::instance()->bluetoothExceptionReadInputStream();
168 }
169
170 ///
```

```
171 /// \brief bluetoothExceptionWriteOutputStream
172 ///
173 static void bluetoothExceptionWriteOutputStream(JNIEnv * /*env*/, jclass /*
        clazz*/)
174 {
175     Bluetooth::instance()->bluetoothExceptionWriteOutputStream();
176 }
177
178 static JNINativeMethod methods[] = {
179     {"bluetoothDeviceFound", "(Ljava/lang/String;)V", (void *)
            bluetoothDeviceFound},
180     {"bluetoothDiscoveryStarted", "()V", (void *)bluetoothDiscoveryStarted},
181     {"bluetoothSearchDone", "()V", (void *)bluetoothSearchDone},
182     {"bluetoothAdapterNotFound", "()V", (void *)bluetoothAdapterNotFound},
183     {"bluetoothDataReceived", "(Ljava/lang/String;)V", (void *)
            bluetoothDataReceived},
184     {"bluetoothExceptionCreateRFCOMM", "()V", (void *)
            bluetoothExceptionCreateRFCOMM},
185     {"bluetoothExceptionJoinThread", "()V", (void *)
            bluetoothExceptionJoinThread},
186     {"bluetoothExceptionInputOutputStream", "()V", (void *)
            bluetoothExceptionInputOutputStream},
187     {"bluetoothExceptionReadInputStream", "()V", (void *)
            bluetoothExceptionReadInputStream},
188     {"bluetoothExceptionWriteOutputStream", "()V", (void *)
            bluetoothExceptionWriteOutputStream}
189 };
190
191 ///
192 /// \brief JNI_OnLoad
193 /// \param vm
194 ///
195 jint JNICALL JNI_OnLoad(JavaVM *vm, void *)
196 {
197     JNIEnv *env;
198     if (vm->GetEnv(reinterpret_cast<void **>(&env), JNI_VERSION_1_4) !=
            JNI_OK)
199         return JNI_FALSE;
200
201     jclass clazz = env->FindClass("org/qtproject/qt5/android/bindings/
            MyActivity");
202     if (env->RegisterNatives(clazz, methods, sizeof(methods) / sizeof(
            methods[0])) < 0)
203         return JNI_FALSE;
204
205     return JNI_VERSION_1_4;
206 }
207
208 //Dummy slots
209 void Bluetooth::serialDataRead()
210 {
211
212 }
213
214 void Bluetooth::write(QByteArray byteArray)
215 {
216     QString string(byteArray);
217
218     //Convert QString to Java string.
219     QAndroidJniObject javaString = QAndroidJniObject::fromString(string);
220
221     //Send data
```

```
222     QAndroidJniObject::callStaticMethod<void>("org/qtproject/qt5/android/
            bindings/MyActivity", "BluetoothWrite", "(Ljava/lang/String;)V",
            javaString.object<jstring>());
223 }
224
225 void Bluetooth::on_couldNotConnect()
226 {
227
228 }
229
230 void Bluetooth::on_connected()
231 {
232
233 }
234
235 void Bluetooth::on_portSettingsFailed()
236 {
237
238 }
239
240 void Bluetooth::on_openPort()
241 {
242
243 }
```

Listing B.8: bluetooth_desktop.cpp

```
1  #include "bluetooth.h"
2  #include <QWidget>
3  #include <QString>
4  #include <QDebug>
5  #include <QSerialPort>
6  #include <QSerialPortInfo>
7  #include <QThread>
8
9  Bluetooth::Bluetooth(QWidget *parent) :
10     QWidget(parent)
11 {
12     //Initialize singleton.
13     singletonInstance = this;
14     bluetoothSelectedDevice = new QString();
15 }
16
17 void Bluetooth::sendData(QString string)
18 {
19     //Send data to serial port.
20     serialPort->write(string.toLatin1(), string.length());
21 }
22
23 void Bluetooth::serialDataRead()
24 {
25     //Get data from serial port.
26     QByteArray readByteData = serialPort->readAll();
27
28     if(readByteData == NULL)
29     {
30         qDebug() << "No data received from port. This might be due to error
               or that the port had no data ready";
31         return;
32     }
33
34     //Convert to string.
35     QString dataString(readByteData);
36
37     //Send to packet class.
38     emit serialDataReady(dataString);
39 }
40
41 void Bluetooth::connect()
42 {
43     //Connect
44     qDebug() << "Qt: Connect";
45 }
46
47 void Bluetooth::discover()
48 {
49     qDebug() << "Qt: Search for COM ports";
50
51     emit clearDiscoveryList();
52
53     ports = new QList<QSerialPortInfo>();
54
55     //Search for available COM ports.
56     *ports = QSerialPortInfo::availablePorts();
57
58     qDebug() << "Number of ports found: " + QString::number(ports->length())
           ;
```

```
59
60      for(int i = 0; i < ports->length(); i++)
61      {
62          QSerialPortInfo temp = ports->at(i);
63          qDebug() << "Description: " + temp.description();
64          qDebug() << "Manufacturer: " + temp.manufacturer();
65          qDebug() << "Port name: " + temp.portName();
66          qDebug() << "System Location: " + temp.systemLocation() + "\n";
67          emit bluetoothDeviceFound(temp.portName() + ": " + temp.description
                ());
68      }
69  }
70
71  void Bluetooth::write(QString string)
72  {
73      QByteArray dataToWrite = string.toLatin1();
74      serialPort->write(dataToWrite);
75  }
76
77  void Bluetooth::write(QByteArray byteArray)
78  {
79      serialPort->write(byteArray);
80  }
81
82  void Bluetooth::deviceSelected()
83  {
84      QStringList comPort;
85
86      comPort = bluetoothSelectedDevice->split(":");
87
88      qDebug() << comPort.at(0);
89
90      //Create COM port object to be used with QSerialPort by providing COM
            port in text.
91      QSerialPortInfo temp(comPort.at(0));
92
93      if(temp.isBusy())
94      {
95          qDebug() << "COM port is already open!";
96          return;
97      }
98
99      //Create serial port object.
100     serialPort = new QSerialPort(temp);
101
102     //Try to connect.
103     if(!serialPort->open(QIODevice::ReadWrite))
104     {
105         qDebug() << "Failed to open port";
106         emit couldNotConnect();
107         return;
108     }
109
110     //Set port settings.
111     if(serialPort->setBaudRate(QSerialPort::Baud115200) &&
112        serialPort->setDataBits(QSerialPort::Data8) &&
113        serialPort->setParity(QSerialPort::NoParity) &&
114        serialPort->setStopBits(QSerialPort::OneStop) &&
115        serialPort->setFlowControl(QSerialPort::NoFlowControl) != true)
116     {
117         qDebug() << "Failed to configure port";
118         emit couldNotConnect();
```

```
119         //emit portSettingsFailed();
120     }
121
122     emit connected();
123
124     QObject::connect(serialPort, SIGNAL(readyRead()), this, SLOT(
            serialDataRead()));
125 }
126
127 void Bluetooth::on_couldNotConnect()
128 {
129     emit couldNotConnect();
130 }
131
132 void Bluetooth::on_connected()
133 {
134     emit connected();
135     QObject::connect(serialPort, SIGNAL(readyRead()), this, SLOT(
            serialDataRead()));
136 }
137
138 void Bluetooth::on_portSettingsFailed()
139 {
140     emit couldNotConnect();
141 }
142
143 void Bluetooth::on_openPort()
144 {
145     //Try to connect 5 times.
146     for(int i = 0; i < 5; i++)
147     {
148
149         if(!serialPort->open(QIODevice::ReadWrite))
150         {
151             qDebug() << "Failed to open port";
152             if(i == 4)
153             {
154                 emit couldNotConnect();
155                 return;
156             }
157         }
158         else
159         {
160             break;
161         }
162     }
163
164     //Set port settings.
165     if(serialPort->setBaudRate(QSerialPort::Baud115200) &&
166        serialPort->setDataBits(QSerialPort::Data8) &&
167        serialPort->setParity(QSerialPort::NoParity) &&
168        serialPort->setStopBits(QSerialPort::OneStop) &&
169        serialPort->setFlowControl(QSerialPort::NoFlowControl) != true)
170     {
171         qDebug() << "Failed to configure port";
172         //emit portSettingsFailed();
173     }
174
175     //emit connected();
176 }
```

Listing B.9: bluetoothdevicelist.h

```
#ifndef BLUETOOTHDEVICELIST_H
#define BLUETOOTHDEVICELIST_H

#include <QWidget>
#include <QListWidget>

class BluetoothDeviceList : public QListWidget
{
    Q_OBJECT
public:
    explicit BluetoothDeviceList(QWidget *parent = 0);

private:

public slots:
    void addDevice(QString device);
    void clearDevices();

private slots:
    void on_deviceList_itemClicked(QListWidgetItem *item);

signals:
    void bluetoothConnect();
    void setSubMenuLabelText(QString text);
    void setSelectedDevice(QString device);
};

#endif // BLUETOOTHDEVICELIST_H
```

Listing B.10: bluetoothdevicelist.cpp

```
1  #include "bluetoothdevicelist.h"
2  #include <QDebug>
3  #include <QListWidget>
4  #include <QMessageBox>
5
6  BluetoothDeviceList::BluetoothDeviceList(QWidget *parent) :
7      QListWidget(parent)
8  {
9      this->setSelectionMode(QAbstractItemView::SingleSelection);
10     connect(this, SIGNAL(itemClicked(QListWidgetItem*)), this, SLOT(
           on_deviceList_itemClicked(QListWidgetItem*)));
11 }
12
13 void BluetoothDeviceList::addDevice(QString device)
14 {
15     qDebug() << "Bluetooth device found and added to list";
16
17     //Add devices to list.
18     this->addItem(device);
19     //new QListWidgetItem(device, ui->deviceList);
20 }
21
22 void BluetoothDeviceList::clearDevices()
23 {
24     qDebug() << "Clear devices list";
25
26     this->clear();
27 }
28
29 void BluetoothDeviceList::on_deviceList_itemClicked(QListWidgetItem *item)
30 {
31     qDebug() << "Device selected: " + item->text();
32
33     //Set selected device in all slots listning.
34     setSelectedDevice(item->text());
35
36     QMessageBox::StandardButton reply;
37     reply = QMessageBox::question(this, "Connect to device", "Do you want to
           connect to: " + item->text() + "?", QMessageBox::Yes | QMessageBox
           ::No);
38
39     if(reply == QMessageBox::Yes)
40     {
41         emit bluetoothConnect();
42     }
43
44     if(reply == QMessageBox::No)
45     {
46         //Do nothing.
47     }
48 }
```

Listing B.11: measurementplots.h

```
#ifndef MEASUREMENTPLOTS_H
#define MEASUREMENTPLOTS_H

#include <QWidget>
#include "QCustomPlot/qcustomplot.h"
#include <QVector>
#include <QMap>

namespace Ui {
class MeasurementPlots;
}

class MeasurementPlots : public QWidget
{
    Q_OBJECT

public:
    explicit MeasurementPlots(QWidget *parent = 0);
    ~MeasurementPlots();

public slots:
    void realtimePlotDataSlot(QString amplitude, QString phase);
    void staticPlotDataSlot(QString frequency, QString amplitude, QString
        phase);
    void onTogglePlotView();

private slots:
    void calculateValuesAndRedrawStaticPlot();
    void changeRealtimePlotTitle(QString frequency);

private:
    Ui::MeasurementPlots *ui;
    void staticPlot();
    void realtimePlot();

    QCPGraph *staticPlotAmplitudeGraph;
    QCPGraph *staticPlotPhaseGraph;
    QCPGraph *realtimePlotAmplitudeGraph;
    QCPGraph *realtimePlotPhaseGraph;
    QMap<QString, QString> *spectroscopyImpedanceData;
    QMap<QString, QString> *spectroscopyPhaseData;
    QCPPlotTitle *title;
};

#endif // MEASUREMENTPLOTS_H
```

Listing B.12: measurementplots.cpp

```
1  #include "measurementplots.h"
2  #include "ui_measurementplots.h"
3  #include <QVector>
4  #include <QMap>
5
6  MeasurementPlots::MeasurementPlots(QWidget *parent) :
7      QWidget(parent),
8      ui(new Ui::MeasurementPlots)
9  {
10     ui->setupUi(this);
11
12     realtimePlot();
13     staticPlot();
14
15     spectroscopyImpedanceData = new QMap<QString, QString>();
16     spectroscopyPhaseData = new QMap<QString, QString>();
17
18     QTimer *staticPlotRedrawTimer = new QTimer(this);
19     connect(staticPlotRedrawTimer, SIGNAL(timeout()), this, SLOT(
           calculateValuesAndRedrawStaticPlot()));
20     staticPlotRedrawTimer->start(1000);
21 }
22
23 MeasurementPlots::~MeasurementPlots()
24 {
25     delete ui;
26 }
27
28 void MeasurementPlots::staticPlotDataSlot(QString frequency, QString
       amplitude, QString phase)
29 {
30     spectroscopyImpedanceData->insert(frequency, amplitude);
31     spectroscopyPhaseData->insert(frequency, phase);
32 }
33
34 void MeasurementPlots::calculateValuesAndRedrawStaticPlot()
35 {
36     QVector<double> frequencyData(30);
37     QVector<double> impedanceData(30);
38     QVector<double> phaseData(30);
39
40     QMap<QString, QString>::const_iterator i = spectroscopyImpedanceData->
           constBegin();
41     while(i != spectroscopyImpedanceData->constEnd())
42     {
43         frequencyData.append(i.key().toDouble());
44         impedanceData.append(i.value().toDouble());
45         i++;
46     }
47
48     QMap<QString, QString>::const_iterator j = spectroscopyPhaseData->
           constBegin();
49     while(j != spectroscopyPhaseData->constEnd())
50     {
51         frequencyData.append(j.key().toDouble());
52         phaseData.append(j.value().toDouble());
53         j++;
54     }
55
56     //Add new data to graphs.
```

```
57      staticPlotAmplitudeGraph->setData(frequencyData, impedanceData);
58      staticPlotPhaseGraph->setData(frequencyData, phaseData);
59
60      //Redraw graph.
61      ui->staticPlot->replot();
62  }
63
64  void MeasurementPlots::realtimePlot()
65  {
66      //Clear standard plot layout.
67      ui->realtimePlot->plotLayout()->clear();
68
69      //Create layout and add to plot widget.
70      QCPLayoutGrid *layout = new QCPLayoutGrid();
71      ui->realtimePlot->plotLayout()->addElement(1, 0, layout);
72
73      title = new QCPPlotTitle(ui->realtimePlot, "Continuous Single Frequency
            Measurement");
74      QFont titleFont;
75      titleFont.setPointSize(20);
76      title->setFont(titleFont);
77      ui->realtimePlot->plotLayout()->addElement(0, 0, title);
78
79      //Create axes rectangle and add it to layout.
80      QCPAxisRect *axesRec = new QCPAxisRect(ui->realtimePlot, false);
81      layout->addElement(0, 0, axesRec);
82
83      //Specify which axes to add.
84      axesRec->addAxis(QCPAxis::atBottom);
85      axesRec->addAxis(QCPAxis::atLeft);
86      axesRec->addAxis(QCPAxis::atRight);
87
88      //Configure x-axis/key-axis.
89      axesRec->axis(QCPAxis::atBottom)->setTickLabelType(QCPAxis::ltDateTime);
90      axesRec->axis(QCPAxis::atBottom)->setDateTimeFormat("hh:mm:ss");
91      axesRec->axis(QCPAxis::atBottom)->setAutoTickStep(false);
92      axesRec->axis(QCPAxis::atBottom)->setTickStep(2);
93      QFont bottomAxisFont = font();
94      bottomAxisFont.setPointSize(10);
95      bottomAxisFont.setBold(true);
96      axesRec->axis(QCPAxis::atBottom)->setLabelFont(bottomAxisFont);
97      axesRec->axis(QCPAxis::atBottom)->setLabel("Time [s]");
98
99      //Configure left/amplitude axis.
100     axesRec->axis(QCPAxis::atLeft)->setRange(100, 100000);
101     axesRec->axis(QCPAxis::atLeft)->setScaleType(QCPAxis::stLogarithmic);
102     axesRec->axis(QCPAxis::atLeft)->setScaleLogBase(10);
103     QFont leftAxisFont = font();
104     leftAxisFont.setPointSize(10);
105     leftAxisFont.setBold(true);
106     axesRec->axis(QCPAxis::atLeft)->setLabelFont(leftAxisFont);
107     axesRec->axis(QCPAxis::atLeft)->setLabel("Impedance Modulus [" + QString
            (QChar(0x03A9)) + "]");
108
109     //Configure right/phase axis.
110     axesRec->axis(QCPAxis::atRight)->setRange(-90, 90);
111     axesRec->axis(QCPAxis::atRight)->setRangeReversed(true);
112
113     axesRec->axis(QCPAxis::atRight)->grid()->setPen(QPen(QBrush(QColor
            (0,0,255,80)),1));
114     axesRec->axis(QCPAxis::atRight)->grid()->setVisible(true);
```

```
115     axesRec->axis(QCPAxis::atRight)->grid()->setSubGridPen(QPen(QBrush(
            QColor(0,0,255,20)),1));
116     axesRec->axis(QCPAxis::atRight)->grid()->setSubGridVisible(true);
117
118     QFont rightAxisFont = font();
119     rightAxisFont.setPointSize(10);
120     rightAxisFont.setBold(true);
121     axesRec->axis(QCPAxis::atRight)->setLabelFont(rightAxisFont);
122     axesRec->axis(QCPAxis::atRight)->setLabel("Phase Angle [degrees]");
123
124     //Create legend.
125     ui->realtimePlot->legend = new QCPLegend();
126
127     //Add legend to graph.
128     ui->realtimePlot->legend->setVisible(true);
129     axesRec->insetLayout()->addElement(ui->realtimePlot->legend, Qt::
            AlignTop | Qt::AlignRight);
130
131     //Create two graph objects. One for amplitude and one for phase.
132     realtimePlotAmplitudeGraph = ui->realtimePlot->addGraph(axesRec->axis(
            QCPAxis::atBottom), axesRec->axis(QCPAxis::atLeft));
133     realtimePlotPhaseGraph = ui->realtimePlot->addGraph(axesRec->axis(
            QCPAxis::atBottom), axesRec->axis(QCPAxis::atRight));
134
135     //Customize amplitude graph.
136     realtimePlotAmplitudeGraph->setPen(QPen(Qt::red));
137     realtimePlotAmplitudeGraph->setLineStyle(QCPGraph::lsLine);
138     realtimePlotAmplitudeGraph->setName("Amplitude");
139
140     //Customize phase graph.
141     realtimePlotPhaseGraph->setPen(QPen(Qt::blue));
142     realtimePlotPhaseGraph->setLineStyle(QCPGraph::lsLine);
143     realtimePlotPhaseGraph->setName("Phase");
144
145     //Customize legend.
146     QFont legendFont = font();
147     legendFont.setBold(true);
148     legendFont.setPointSize(10);
149     ui->realtimePlot->legend->item(0)->setFont(legendFont);
150     ui->realtimePlot->legend->item(1)->setFont(legendFont);
151     ui->realtimePlot->legend->item(0)->setTextColor(QColor(Qt::black));
152     ui->realtimePlot->legend->item(1)->setTextColor(QColor(Qt::black));
153 }
154
155 void MeasurementPlots::realtimePlotDataSlot(QString amplitude, QString phase
        )
156 {
157     //Previous time stamp.
158     static double previousTime = 0;
159
160     //Get current date/time in millisecond resolution.
161     double currentTime = QDateTime::currentDateTime().toMSecsSinceEpoch()
            /1000.0;
162
163     //Only update every 50 ms at most.
164     if (currentTime - previousTime > 0.05)
165     {
166         double amplitudeData = amplitude.toDouble();
167         double phaseData = phase.toDouble();
168
169         //Add new amplitude and phase data.
170         realtimePlotAmplitudeGraph->addData(currentTime, amplitudeData);
```

```
171         realtimePlotPhaseGraph->addData(currentTime, phaseData);
172
173         //Remove data that is out of visible range.
174         realtimePlotAmplitudeGraph->removeDataBefore(currentTime - 40);
175         realtimePlotPhaseGraph->removeDataBefore(currentTime - 40);
176
177         previousTime = currentTime;
178     }
179
180     // make key axis range scroll with the data (at a constant range size of
          8):
181     realtimePlotAmplitudeGraph->keyAxis()->setRange(currentTime + 0.25, 40,
          Qt::AlignRight);
182     ui->realtimePlot->replot();
183 }
184
185 void MeasurementPlots::staticPlot()
186 {
187     //Clear standard plot layout.
188     ui->staticPlot->plotLayout()->clear();
189
190     //Create layout and add to plot widget.
191     QCPLayoutGrid *layout = new QCPLayoutGrid();
192     ui->staticPlot->plotLayout()->addElement(1, 0, layout);
193
194     QCPPlotTitle *title = new QCPPlotTitle(ui->staticPlot, "Frequency vs
          Impedance Modulus and Phase");
195     QFont titleFont;
196     titleFont.setPointSize(20);
197     title->setFont(titleFont);
198     ui->staticPlot->plotLayout()->addElement(0, 0, title);
199
200     //Create axes rectangle and add it to layout.
201     QCPAxisRect *axesRec = new QCPAxisRect(ui->staticPlot, false);
202     layout->addElement(0, 0, axesRec);
203
204     //Specify which axes to add.
205     axesRec->addAxis(QCPAxis::atBottom);
206     axesRec->addAxis(QCPAxis::atLeft);
207     axesRec->addAxis(QCPAxis::atRight);
208
209     //Configure x-axis/key-axis.
210     axesRec->axis(QCPAxis::atBottom)->setTickLabelType(QCPAxis::ltNumber);
211     axesRec->axis(QCPAxis::atBottom)->setScaleType(QCPAxis::stLogarithmic);
212     axesRec->axis(QCPAxis::atBottom)->setScaleLogBase(10);
213     axesRec->axis(QCPAxis::atBottom)->setRange(1000, 1000000);
214     QFont bottomAxisFont = font();
215     bottomAxisFont.setPointSize(10);
216     bottomAxisFont.setBold(true);
217     axesRec->axis(QCPAxis::atBottom)->setLabelFont(bottomAxisFont);
218     axesRec->axis(QCPAxis::atBottom)->setLabel("Frequency [Hz]");
219
220     //Configure left/amplitude axis.
221     axesRec->axis(QCPAxis::atBottom)->setTickLabelType(QCPAxis::ltNumber);
222     axesRec->axis(QCPAxis::atLeft)->setScaleType(QCPAxis::stLogarithmic);
223     axesRec->axis(QCPAxis::atLeft)->setScaleLogBase(10);
224     axesRec->axis(QCPAxis::atLeft)->setRange(100, 100000);
225     QFont leftAxisFont = font();
226     leftAxisFont.setPointSize(10);
227     leftAxisFont.setBold(true);
228     axesRec->axis(QCPAxis::atLeft)->setLabelFont(leftAxisFont);
```

```
229     axesRec->axis(QCPAxis::atLeft)->setLabel("Impedance Modulus [" + QString
            (QChar(0x03A9)) + "]");
230
231     //Configure right/phase axis.
232     //axesRec->axis(QCPAxis::atRight)->setRange(-180.1, 180.1);
233     axesRec->axis(QCPAxis::atRight)->setRange(-90, 90);
234     axesRec->axis(QCPAxis::atRight)->setRangeReversed(true);
235     axesRec->axis(QCPAxis::atRight)->grid()->setPen(QPen(QBrush(QColor
            (0,0,255,80)),1));
236     axesRec->axis(QCPAxis::atRight)->grid()->setVisible(true);
237     axesRec->axis(QCPAxis::atRight)->grid()->setSubGridPen(QPen(QBrush(
            QColor(0,0,255,20)),1));
238     axesRec->axis(QCPAxis::atRight)->grid()->setSubGridVisible(true);
239     QFont rightAxisFont = font();
240     rightAxisFont.setPointSize(10);
241     rightAxisFont.setBold(true);
242     axesRec->axis(QCPAxis::atRight)->setLabelFont(rightAxisFont);
243     axesRec->axis(QCPAxis::atRight)->setLabel("Phase Angle [degrees]");
244
245     //Create legend.
246     ui->staticPlot->legend = new QCPLegend();
247
248     //Add legend to graph.
249     ui->staticPlot->legend->setVisible(true);
250     axesRec->insetLayout()->addElement(ui->staticPlot->legend, Qt::AlignTop
            | Qt::AlignRight);
251
252     //Create two graph objects. One for amplitude and one for phase.
253     staticPlotAmplitudeGraph = ui->staticPlot->addGraph(axesRec->axis(
            QCPAxis::atBottom), axesRec->axis(QCPAxis::atLeft));
254     staticPlotPhaseGraph = ui->staticPlot->addGraph(axesRec->axis(QCPAxis::
            atBottom), axesRec->axis(QCPAxis::atRight));
255
256     //Customize amplitude graph.
257     staticPlotAmplitudeGraph->setPen(QPen(Qt::red));
258     staticPlotAmplitudeGraph->setLineStyle(QCPGraph::lsLine);
259     staticPlotAmplitudeGraph->setName("Amplitude");
260
261     //Customize phase graph.
262     staticPlotPhaseGraph->setPen(QPen(Qt::blue));
263     staticPlotPhaseGraph->setLineStyle(QCPGraph::lsLine);
264     staticPlotPhaseGraph->setName("Phase");
265
266     //Customize legend.
267     QFont legendFont = font();
268     legendFont.setBold(true);
269     legendFont.setPointSize(10);
270     ui->staticPlot->legend->item(0)->setFont(legendFont);
271     ui->staticPlot->legend->item(1)->setFont(legendFont);
272     ui->staticPlot->legend->item(0)->setTextColor(QColor(Qt::black));
273     ui->staticPlot->legend->item(1)->setTextColor(QColor(Qt::black));
274 }
275
276 void MeasurementPlots::onTogglePlotView()
277 {
278     if(ui->staticPlot->isVisible() && ui->realtimePlot->isVisible())
279     {
280         ui->staticPlot->setVisible(false);
281     }
282     else if(!ui->staticPlot->isVisible() && ui->realtimePlot->isVisible())
283     {
284         ui->staticPlot->setVisible(true);
```

```
285         ui->realtimePlot->setVisible(false);
286     }
287     else if(ui->staticPlot->isVisible() && !ui->realtimePlot->isVisible())
288     {
289         ui->realtimePlot->setVisible(true);
290     }
291 }
292
293 void MeasurementPlots::changeRealtimePlotTitle(QString frequency)
294 {
295     title->setText("Continuous Single Frequency Measurement @ " + frequency
            + " Hz");
296 }
```

Listing B.13: menuEnums.h

```
#ifndef MENUENUMS_H
#define MENUENUMS_H

typedef enum
{
    DEVICE_LIST = 0,
    MEASUREMENT_PLOT,
    MEASUREMENT_SETTINGS_REALTIME_PLOT,
    MEASUREMENT_SETTINGS_STATIC_PLOT,
    OTHER
} MAIN_MENU_t;

typedef enum
{
    PLOT_SELECTOR = 0
} SUB_MENU_t;

#endif // MENUENUMS_H
```

Listing B.14: messages.h

```
#ifndef MESSAGES_H
#define MESSAGES_H

enum MessageEnumMeasurementSettings
{
    REALTIME_PLOT                       = 0,
    STATIC_PLOT                         = 1,
    ENABLE                              = 1,
    DISABLE                             = 0,
    CONTINUOUS_SINGLE_FREQUENCY_MODE    = 1,
    QUICK_FREQUENCY_SWEEP_MODE          = 2,
    FULL_FREQUENCY_SWEEP_MODE           = 3,
    RESISTANCE_AND_REACTANCE            = 1,
    CONDUCTANCE_AND_SUSEPTANCE          = 2,
    MODULUS_AND_PHASE                   = 3,
    TECHNIQUE_NORMAL                    = 0,
    TECHNIQUE_4FS                       = 1
};

typedef struct MessageMeasurementSettings
{
    unsigned int realtimePlotEnable     : 1;
    unsigned int spectroscopyPlotEnable : 1;
    unsigned int realtimePlotParameters : 2;
    unsigned int technique              : 1;
    unsigned int frequency              : 20;
    unsigned int calibratePhase         : 1;
    unsigned int systemMode             : 4;
    unsigned int reserved               : 3;
} MessageMeasurementSettings;

enum MessageIDOutgoing
{
    PREAMBLE                        = 0xAA,
    MEASUREMENT_SETTINGS            = 0x01,

    MESSAGE_END                     = 0x0A
};

enum MessageIDIncoming
{
    NOT_VALID                           = 0,
    DDS                                 = 1,
    CONTINUOUS_SINGLE_FREQUENCY_4FS     = 2,
    FULL_FREQUENCY_SWEEP_4FS            = 3,
    QUICK_FREQUENCY_SWEEP_4FS           = 4,
    CONTINUOUS_SINGLE_FREQUENCY_NORMAL  = 5,
    FULL_FREQUENCY_SWEEP_NORMAL         = 6,
    QUICK_FREQUENCY_SWEEP_NORMAL        = 7,
    SYSTEM_STATUS                       = 8,
    CURRENT                             = 9,
    RESISTANCE_REACTANCE                = 10,
    CONDUCTANCE_SUSCEPTANCE             = 11,
    IMPEDANCE_ADMITTANCE                = 12,
    SPECTROSCOPY_DATA                   = 13,
    CSFM_FREQUENCY                      = 14,
    EGG_STATE                           = 15,

    CALIBRATION_STATUS                  = 98,
    HEARTBEAT                           = 99
```

```
61  };
62
63  enum MessageIDIncomingLength
64  {
65      DDS_MSG_LENGTH                                  = 2,
66      CONTINUOUS_SINGLE_FREQUENCY_4FS_MSG_LENGTH      = 3,
67      FULL_FREQUENCY_SWEEP_4FS_MSG_LENGTH             = 0,
68      QUICK_FREQUENCY_SWEEP_4FS_MSG_LENGTH            = 0,
69      CONTINUOUS_SINGLE_FREQUENCY_NORMAL_MSG_LENGTH   = 0,
70      FULL_FREQUENCY_SWEEP_NORMAL_MSG_LENGTH          = 0,
71      QUICK_FREQUENCY_SWEEP_NORMAL_MSG_LENGTH         = 0,
72      SYSTEM_STATUS_MSG_LENGTH                        = 2,
73      CURRENT_MSG_LENGTH                              = 2,
74      RESISTANCE_REACTANCE_MSG_LENGTH                 = 3,
75      CONDUCTANCE_SUSCEPTANCE_MSG_LENGTH              = 3,
76      IMPEDANCE_ADMITTANCE_MSG_LENGTH                 = 3,
77      SPECTROSCOPY_DATA_MSG_LENGTH                    = 4,
78      CSFM_FREQUENCY_MSG_LENGTH                       = 2,
79      EGG_STATE_MSG_LENGTH                            = 2,
80
81      CALIBRATION_STATUS_MSG_LENGTH                   = 2,
82      HEARTBEAT_MSG_LENGTH                            = 2
83  };
84
85  #endif // MESSAGES_H
```

Listing B.15: receive.h

```
#ifndef RECEIVE_H
#define RECEIVE_H

#include <QObject>

class Receive : public QObject
{
    Q_OBJECT
public:
    explicit Receive(QObject *parent = 0);

private:
    QString getMsg();
    QString parseMessage();
    bool isMsgChecksumOK(QString msg);
    void errorCorrectionCode();
    void incrementMessageCounter();
    QString * buffer;
    QStringList * messages;
    unsigned int msgCounter;

signals:
    void messageCounterValueChanged();
    void amplitudeAndPhaseDataReceived(QString amplitude, QString phase);
    void frequencyAmplitudePhaseDataReceived(QString frequency, QString
        amplitude, QString phase);
    void resistanceReactance(QString resistance, QString Reactance);
    void conductanceSusceptance(QString conductance, QString Susceptance);
    void impedanceAdmittance(QString impedance, QString admittance);
    void spectropscopyDataUpdate(QString frequency, QString resistance,
        QString reactance);
    void measurementFrequencyChanged(QString frequency);
    void eggState(QString state);
    void heartbeat();
    void calibrationStatus(QString state);

public slots:
    void addData(QString data);

private slots:
    void messageReady();
};

#endif // RECEIVE_H
```

Listing B.16: receive.cpp

```
#include "receive.h"
#include <QDebug>
#include <QString>
#include <QStringList>
#include "messages.h"

Receive::Receive(QObject *parent) :
    QObject(parent)
{
    buffer = new QString();
    messages = new QStringList();
    msgCounter = 0;

    connect(this, SIGNAL(messageCounterValueChanged()), this, SLOT(
        messageReady()));
}

void Receive::addData(QString data)
{
    //Add data to buffer.
    *buffer += data;

    //Check if this data completes a message.
    QStringList temp = buffer->split("\n");

    if(temp.length() < 2)
    {
        return;
    }

    //Add completed messages and increase message counter.
    for(int i = 0; i < temp.length() - 1; i++)
    {
        messages->append(temp.at(i));
        incrementMessageCounter();
    }

    //Add uncompleted messages to buffer.
    *buffer = temp.at(temp.length() - 1);
}

void Receive::incrementMessageCounter()
{
    msgCounter++;
    messageCounterValueChanged();
}

void Receive::messageReady()
{
    while (msgCounter > 0)
    {
        parseMessage();
    }
}

QString Receive::getMsg()
{
    if(messages->isEmpty())
    {
        return "";
```

```
60      }
61      msgCounter--;
62      return messages->takeFirst();
63  }
64
65  QString Receive::parseMessage()
66  {
67      //Get message.
68      QString msg = getMsg();
69
70      //Check if message is empty.
71      if(msg == NULL)
72      {
73          return "";
74      }
75
76      //Check if message is ok.
77      if(!isMsgChecksumOK(msg))
78      {
79          return "";
80      }
81
82      QStringList msgFields = msg.split(":");
83
84      if(msgFields.at(0) == "!!!") qDebug() << msgFields.at(0) + msgFields.at
            (1);
85
86      switch (msgFields.at(0).toInt()) {
87
88      case DDS:
89          if(msgFields.length() != DDS_MSG_LENGTH)
90          {
91              QString temp;
92
93              temp += "Message corrupted. DDS: ";
94
95              for(int i = 0; i < msgFields.length(); i++)
96              {
97                  if(i==0) temp += "Field 0 (MSG type): " + msgFields.at(0);
98                  if(i==1) temp += "Field 1 (Amplitude): " + msgFields.at(1);
99                  if(i>=2) temp += "Field " + QString::number(i) + " (unknown)
                        : " + msgFields.at(i);
100             }
101
102             qDebug() << temp;
103             return "";
104         }
105
106         qDebug() << "DDS amplitude: " + msgFields.at(1);
107         break;
108
109     case CONTINUOUS_SINGLE_FREQUENCY_4FS:
110         if(msgFields.length() != CONTINUOUS_SINGLE_FREQUENCY_4FS_MSG_LENGTH)
111         {
112             QString temp;
113
114             temp += "Message corrupted. CSF4FS: ";
115
116             for(int i = 0; i < msgFields.length(); i++)
117             {
118                 if(i==0) temp += "Field 0 (MSG type): " + msgFields.at(0);
119                 if(i==1) temp += "Field 1 (Amplitude): " + msgFields.at(1);
```

```
120                 if(i==2) temp += "Field 2 (Phase): " + msgFields.at(2);
121                 if(i>=3) temp += "Field " + QString::number(i) + " (unknown)
                        : " + msgFields.at(i);
122             }
123
124             qDebug() << temp;
125
126             return "";
127         }
128
129         emit amplitudeAndPhaseDataReceived(msgFields.at(1), msgFields.at(2))
                ;
130         break;
131
132     case FULL_FREQUENCY_SWEEP_4FS:
133
134         break;
135
136     case QUICK_FREQUENCY_SWEEP_4FS:
137
138         break;
139
140     case CONTINUOUS_SINGLE_FREQUENCY_NORMAL:
141
142         break;
143
144     case FULL_FREQUENCY_SWEEP_NORMAL:
145
146         break;
147
148     case QUICK_FREQUENCY_SWEEP_NORMAL:
149
150         break;
151
152     case SYSTEM_STATUS:
153         if(msgFields.length() != SYSTEM_STATUS_MSG_LENGTH)
154         {
155             QString temp;
156
157             temp += "Message corrupted. SYSTEM_STATUS: ";
158
159             for(int i = 0; i < msgFields.length(); i++)
160             {
161                 if(i==0) temp += "Field 0 (MSG type): " + msgFields.at(0);
162                 if(i==1) temp += "Field 1 (System Status): " + msgFields.at
                        (1);
163                 if(i>=2) temp += "Field " + QString::number(i) + " (unknown)
                        : " + msgFields.at(i);
164             }
165
166             qDebug() << temp;
167             return "";
168         }
169
170         qDebug() << "System Status: " + msgFields.at(1);
171         break;
172
173     case CURRENT:
174         if(msgFields.length() != CURRENT_MSG_LENGTH)
175         {
176             QString temp;
177
```

```
178             temp += "Message corrupted. CURRENT: ";
179
180             for(int i = 0; i < msgFields.length(); i++)
181             {
182                 if(i==0) temp += "Field 0 (MSG type): " + msgFields.at(0);
183                 if(i==1) temp += "Field 1 (Current): " + msgFields.at(1);
184                 if(i>=2) temp += "Field " + QString::number(i) + " (unknown)
                        : " + msgFields.at(i);
185             }
186
187             qDebug() << temp;
188             return "";
189         }
190
191         qDebug() << "Current (mA): " + msgFields.at(1);
192         //emit current(msgFields.at(1));
193         break;
194
195     case RESISTANCE_REACTANCE:
196         if(msgFields.length() != RESISTANCE_REACTANCE_MSG_LENGTH)
197         {
198             break;
199         }
200
201         emit resistanceReactance(msgFields.at(1), msgFields.at(2));
202
203         break;
204
205     case CONDUCTANCE_SUSCEPTANCE:
206         if(msgFields.length() != CONDUCTANCE_SUSCEPTANCE_MSG_LENGTH)
207         {
208             break;
209         }
210
211         emit conductanceSusceptance(msgFields.at(1), msgFields.at(2));
212
213         break;
214
215     case IMPEDANCE_ADMITTANCE:
216         if(msgFields.length() != IMPEDANCE_ADMITTANCE_MSG_LENGTH)
217         {
218             qDebug() << "Invalid real time data recieved!";
219             break;
220         }
221
222         emit impedanceAdmittance(msgFields.at(1), msgFields.at(2));
223
224         break;
225
226     case SPECTROSCOPY_DATA:
227         if(msgFields.length() != SPECTROSCOPY_DATA_MSG_LENGTH)
228         {
229             qDebug() << "Invalid spectroscopy data received!";
230             break;
231         }
232
233         emit spectropscopyDataUpdate(msgFields.at(1), msgFields.at(2),
                msgFields.at(3));
234
235         break;
236
237     case CSFM_FREQUENCY:
```

```
238         if(msgFields.length() != CSFM_FREQUENCY_MSG_LENGTH)
239         {
240             qDebug() << "Invalid CSFM frequency received!";
241         }
242
243         emit measurementFrequencyChanged(msgFields.at(1));
244
245         break;
246
247     case EGG_STATE:
248         if(msgFields.length() != EGG_STATE_MSG_LENGTH)
249         {
250             qDebug() << "Invalid Egg state received!";
251         }
252
253         emit eggState(msgFields.at(1));
254
255         break;
256
257     case HEARTBEAT:
258         if(msgFields.length() != HEARTBEAT_MSG_LENGTH)
259         {
260             qDebug() << "Invalid heart beat msg recieved";
261         }
262
263         emit heartbeat();
264
265         break;
266
267     case CALIBRATION_STATUS:
268         if(msgFields.length() != CALIBRATION_STATUS_MSG_LENGTH)
269         {
270             qDebug() << "Invalid calibration status msg recieved";
271         }
272
273         emit calibrationStatus(msgFields.at(1));
274
275         break;
276
277
278     default:
279         qDebug() << msg;
280         break;
281     }
282
283     return msg;
284 }
285
286 bool Receive::isMsgChecksumOK(QString msg)
287 {
288     //Checksum algorithm.
289     return true;
290 }
291
292 void Receive::errorCorrectionCode()
293 {
294
295 }
```

Listing B.17: send.h

```
#ifndef SEND_H
#define SEND_H

#include <QObject>
#include "messages.h"

class Send : public QObject
{
    Q_OBJECT
public:
    explicit Send(QObject *parent = 0);

signals:
    void sendMeasurementSettings(QByteArray msg);

public slots:
    void measurementSettings(MessageMeasurementSettings msg);
};

#endif // SEND_H
```

Listing B.18: send.cpp

```
1  #include "send.h"
2  #include "messages.h"
3  #include <QtEndian>
4
5  Send::Send(QObject *parent) :
6      QObject(parent)
7  {
8  }
9
10 void Send::measurementSettings(MessageMeasurementSettings msg)
11 {
12     quint8 * tempPtr = (quint8 *) &msg;
13     unsigned int *temp = (unsigned int *) &msg;
14
15     //Construct message with preamble and message ID.
16     QByteArray tempByteArray;
17     tempByteArray.append(PREAMBLE);
18     tempByteArray.append(MEASUREMENT_SETTINGS);
19     tempByteArray.append(tempPtr[0]);
20     tempByteArray.append(tempPtr[1]);
21     tempByteArray.append(tempPtr[2]);
22     tempByteArray.append(tempPtr[3]);
23     tempByteArray.append(MESSAGE_END);
24
25     emit sendMeasurementSettings(tempByteArray);
26 }
```

## B.3 Code: Java

.

Listing B.19: MyActivity.java

```
1  package org.qtproject.qt5.android.bindings;
2
3  //Qt imports
4  import org.qtproject.qt5.android.bindings.QtApplication;
5  import org.qtproject.qt5.android.bindings.QtActivity;
6
7  //Java imports
8  import java.io.IOException;
9  import java.io.InputStream;
10 import java.io.OutputStream;
11 import java.lang.Byte;
12 import java.lang.String;
13 import java.util.Queue;
14 import java.util.UUID;
15
16 //Android imports
17 import android.app.Activity;
18 import android.bluetooth.BluetoothAdapter;
19 import android.bluetooth.BluetoothDevice;
20 import android.bluetooth.BluetoothSocket;
21 import android.content.BroadcastReceiver;
22 import android.content.Context;
23 import android.content.Intent;
24 import android.content.IntentFilter;
25 import android.os.Bundle;
26 import android.os.Handler;
27 import android.os.Message;
28 import android.util.Log;
29
30 public class MyActivity extends QtActivity
31 {
32         ///////////////
33         // Objects. //
34         ///////////////
35
36         private static MyActivity singletonInstance;
37         private BluetoothAdapter bluetoothAdapter;
38         private BluetoothDevice device;
39         private BluetoothSocket bluetoothSocket = null;
40         private BluetoothConnectThread bluetoothConnectThread;
41         private InputStream inputStream;
42         private OutputStream outputStream;
43         private byte[] bytesSend;
44
45         ////////////////////////////
46         // Global data fields. //
47         ////////////////////////////
48
49         private String[] devicesFound;
50
51         ////////////////
52         // Constants. //
53         ////////////////
54
55         private static final UUID uuid = UUID.fromString("
```

```
            00001101-0000-1000-8000-00805F9B34FB");
56          private static final int ENABLE_BLUETOOTH_REQUEST = 1;
57          private static final int maxDevicesFound = 25;
58          private static final int outgoingMsgQueueSize = 100;
59          private static final int incomingMsgQueueSize = 1024;
60
61          public MyActivity()
62          {
63              singletonInstance = this;
64              devicesFound = new String[maxDevicesFound];
65              bytesSend = new byte[outgoingMsgQueueSize];
66          }
67
68          ////////////////////////////////////
69          // Reimplemented superclass methods. //
70          ////////////////////////////////////
71
72          //This method is run when an object of this class is instantiated.
73          @Override
74          public void onCreate(Bundle savedInstanceState)
75          {
76              super.onCreate(savedInstanceState);
77          }
78
79          //This method is run before an object of this class gets deleted.
80          @Override
81          public void onDestroy()
82          {
83              super.onDestroy();
84              Log.d(QtApplication.QtTAG, "JAVA: onDestroy triggered!");
85              unregisterReceiver(BluetoothReceiver);
86          }
87
88          //This method captures events, often user generated.
89          @Override
90          protected void onActivityResult(int requestCode, int resultCode,
                Intent data)
91          {
92              Log.d(QtApplication.QtTAG, "JAVA: onActivityResult entered");
93
94              switch (requestCode)
95              {
96                  case ENABLE_BLUETOOTH_REQUEST:
97                      if(resultCode == RESULT_OK)
98                      {
99                          Log.d(QtApplication.QtTAG, "JAVA: User accepted to
                                enable Bluetooth\nJAVA: Bluetooth is on");
100                     }
101                     else if(resultCode == RESULT_CANCELED)
102                     {
103                         Log.d(QtApplication.QtTAG, "JAVA: User declined to
                                enable Bluetooth");
104                     }
105                     break;
106
107                 default:
108                     break;
109             }
110         }
111
112         //////////////////////
113         // Native methods. //
```

```
114         ////////////////////
115
116         //Native methods are used to signal the C++ side of the application
                and are used as Qt signals.
117         private static native void bluetoothDeviceFound(String device);
118         private static native void bluetoothDiscoveryStarted();
119         private static native void bluetoothSearchDone();
120         private static native void bluetoothAdapterNotFound();
121         private static native void bluetoothDataReceived(String data);
122         private static native void bluetoothExceptionCreateRFCOMM();
123         private static native void bluetoothExceptionJoinThread();
124         private static native void bluetoothExceptionInputOutputStream();
125         private static native void bluetoothExceptionReadInputStream();
126         private static native void bluetoothExceptionWriteOutputStream();
127
128         ///////////////
129         // Methods. //
130         ///////////////
131
132         //Get used UUID.
133         public static String BluetoothGetUUID() {
134             return uuid.toString();
135         }
136
137         //Check if bluetooth hardware is available on device and ask user to
                 enable bluetooth if not already enabled.
138         public static void BluetoothConnect()
139         {
140             Log.d(QtApplication.QtTAG, "JAVA: MyActivity.Bluetooth.connect()
                    ");
141             singletonInstance.bluetoothAdapter = BluetoothAdapter.
                    getDefaultAdapter();
142             if(singletonInstance.bluetoothAdapter == null)
143             {
144                 Log.d(QtApplication.QtTAG, "JAVA: Bluetooth adapter is not
                        found");
145
146                 //Alert user that bluetooth hardware not found on this
                        device.
147                 bluetoothAdapterNotFound();
148                 return;
149             }
150
151             if(!singletonInstance.bluetoothAdapter.isEnabled())
152             {
153                 Log.d(QtApplication.QtTAG, "JAVA: Bluetooth is off");
154                 Intent enableBluetoothIntent = new Intent(BluetoothAdapter.
                        ACTION_REQUEST_ENABLE);
155                 singletonInstance.startActivityForResult(
                        enableBluetoothIntent, ENABLE_BLUETOOTH_REQUEST);
156             }
157             else
158             {
159                 Log.d(QtApplication.QtTAG, "JAVA: Bluetooth is already on");
160             }
161         }
162
163         //Starts searching for nearby bluetooth devices.
164         public static void BluetoothStartDiscovery()
165         {
166             //Add "bluetooth device found" filter to bluetooth receiver.
167             IntentFilter filter = new IntentFilter(BluetoothDevice.
```

```
                ACTION_FOUND);
168             singletonInstance.registerReceiver(singletonInstance.
                    BluetoothReceiver, filter);
169
170             //Add "bluetooth device discovery finished" filter to bluetooth
                    receiver.
171             filter = new IntentFilter(BluetoothAdapter.
                    ACTION_DISCOVERY_FINISHED);
172             singletonInstance.registerReceiver(singletonInstance.
                    BluetoothReceiver, filter);
173
174             //If already discovering, stop.
175             BluetoothCancelDiscovery();
176
177             //Notify C++ that discovery has started.
178             bluetoothDiscoveryStarted();
179
180             //Start discovery
181             singletonInstance.bluetoothAdapter.startDiscovery();
182         }
183
184         public static void BluetoothCancelDiscovery()
185         {
186             //If already discovering, stop.
187             if (singletonInstance.bluetoothAdapter.isDiscovering()) {
188                 singletonInstance.bluetoothAdapter.cancelDiscovery();
189             }
190         }
191
192         //Connect to device with macAddress.
193         public static void BluetoothConnectToDevice(String macAddress)
194         {
195             Log.d(QtApplication.QtTAG, "MAC address for received device: " +
                     macAddress);
196
197             //Checks if MAC address is in a valid format
198             if(!BluetoothAdapter.checkBluetoothAddress(macAddress)) {
199                 singletonInstance.finish();
200             }
201
202             //Get a device object from MAC address
203             singletonInstance.device = singletonInstance.bluetoothAdapter.
                    getRemoteDevice(macAddress);
204
205             BluetoothSocket tempSocket = null;
206
207             //Try to connect a bluetooth socket with the device
208             try {
209                 //SPP(Serial Port Protocol) UUID. Create RFCOMM bluetooth
                        socket.
210                     tempSocket = singletonInstance.device.
                            createRfcommSocketToServiceRecord(uuid);
211             } catch (IOException e) {
212                 bluetoothExceptionCreateRFCOMM();
213             }
214
215             //Assign the successfully received tempSocket to bluetoothSocket
                    .
216             singletonInstance.bluetoothSocket = tempSocket;
217
218             singletonInstance.bluetoothConnectThread = new
                    BluetoothConnectThread(singletonInstance.bluetoothSocket);
```

```
219             singletonInstance.bluetoothConnectThread.start();
220
221             try {
222                 //Wait for thread that tries to connect to bluetoothsocket.
223                 singletonInstance.bluetoothConnectThread.join();
224             } catch (InterruptedException e) {
225                 bluetoothExceptionJoinThread();
226             }
227
228             try {
229                     singletonInstance.inputStream = singletonInstance.
                            bluetoothSocket.getInputStream();
230                     singletonInstance.outputStream = singletonInstance.
                            bluetoothSocket.getOutputStream();
231             } catch (IOException e) {
232                 bluetoothExceptionInputOutputStream();
233             }
234
235             singletonInstance.BluetoothStartDataTransferThread();
236         }
237
238         //Bluetooth receive data thread.
239         private void BluetoothStartDataTransferThread()
240         {
241             Thread bluetoothDataTransferThread = new Thread() {
242                 public void run() {
243                     byte[] buffer = new byte[incomingMsgQueueSize];
244                     int bytesRead, i;
245                     String bufferString = "";
246                     Bundle bundle = new Bundle();
247
248                     //Listen for incoming data.
249                     while (true) {
250                         try {
251                         //Read incoming data stream.
252                         bytesRead = inputStream.read(buffer, 0,
                                incomingMsgQueueSize);
253                         } catch (IOException e) {
254                             bluetoothExceptionReadInputStream();
255                             break;
256                         }
257
258                         if( (bytesRead > 0) && (buffer.length != 0) ) {
259                             //Get data from buffer to string.
260                             bufferString = new String(buffer, 0, bytesRead);
261
262                             if(bufferString == null) {
263                                 continue;
264                             }
265
266                             //Create Inter Thread Communication message.
267                             Message msg = Message.obtain();
268
269                             //Add data to message.
270                             bundle.putString("ReceivedBluetoothData",
                                    bufferString);
271                             msg.setData(bundle);
272
273                             //Send message to handler.
274                             BluetoothReceiveMessageHandler.sendMessage(msg);
275                         }
276                     }
```

```
277                 }
278             };
279 
280             //Start thread.
281             bluetoothDataTransferThread.start();
282         }
283 
284         //Received bluetooth data handler.
285         private Handler BluetoothReceiveMessageHandler = new Handler() {
286             @Override
287             public void handleMessage(Message msg) {
288                 //Get message data.
289                 Bundle bundle = msg.getData();
290 
291                 //Send string to C++.
292                 String temp = bundle.getString("ReceivedBluetoothData");
293                 if(temp != null)
294                 {
295                     bluetoothDataReceived(temp);
296                 }
297             }
298         };
299 
300         //Write data to connected bluetooth device.
301         public static void BluetoothWrite(String string) {
302             Log.d(QtApplication.QtTAG, "JAVA: Text to send: " + string);
303             singletonInstance.bytesSend = string.getBytes();
304             try {
305                 singletonInstance.outputStream.write(singletonInstance.
                        bytesSend);
306             } catch (IOException e) {
307                 bluetoothExceptionWriteOutputStream();
308             }
309         }
310 
311         ////////////////////////
312         // Broadcast Receivers. //
313         ////////////////////////
314 
315         private final BroadcastReceiver BluetoothReceiver = new
                BroadcastReceiver() {
316             public void onReceive(Context context, Intent intent) {
317                 String action = intent.getAction();
318 
319                 if (BluetoothDevice.ACTION_FOUND.equals(action)) {
320                     //New device discovered.
321 
322                     //Get the bluetooth device object.
323                     BluetoothDevice device = intent.getParcelableExtra(
                            BluetoothDevice.EXTRA_DEVICE);
324 
325                     //Send device information to C++. 17 last characters is
                            MAC address.
326                     bluetoothDeviceFound(device.getName() + device.
                            getAddress());
327                 }
328                 else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(
                        action)) {
329                     //Notify C++ that discovery is done.
330                     bluetoothSearchDone();
331                 }
332             }
```

```
333         };
334 }
335
336 class BluetoothConnectThread extends Thread {
337     private final BluetoothSocket bluetoothSocket;
338
339     public BluetoothConnectThread(BluetoothSocket bluetoothSocket) {
340         this.bluetoothSocket = bluetoothSocket;
341     }
342
343     public void run() {
344         try {
345             //Try to connect to bluetoothSocket.
346             bluetoothSocket.connect();
347         } catch (IOException connectException) {
348
349             //If unable to connect, try to close socket and return.
350                 try {
351                 bluetoothSocket.close();
352             } catch (IOException closeException) {}
353
354             return;
355         }
356     }
357 }
```

## B.4 Code: XML

.

Listing B.20: mainwindow.ui

```
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow" >
  <property name="geometry" >
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle" >
   <string>MainWindow</string>
  </property>
  <widget class="QMenuBar" name="menuBar" />
  <widget class="QToolBar" name="mainToolBar" />
  <widget class="QWidget" name="centralWidget" />
  <widget class="QStatusBar" name="statusBar" />
 </widget>
 <layoutDefault spacing="6" margin="11" />
 <pixmapfunction></pixmapfunction>
 <resources/>
 <connections/>
</ui>
```

Listing B.21: measurementplots.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MeasurementPlots</class>
 <widget class="QWidget" name="MeasurementPlots">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1148</width>
    <height>728</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Form</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="QCustomPlot" name="realtimePlot" native="true"/>
   </item>
   <item>
    <widget class="QCustomPlot" name="staticPlot" native="true"/>
   </item>
  </layout>
 </widget>
 <customwidgets>
  <customwidget>
   <class>QCustomPlot</class>
   <extends>QWidget</extends>
   <header>QCustomPlot/qcustomplot.h</header>
   <container>1</container>
  </customwidget>
 </customwidgets>
 <resources/>
 <connections/>
</ui>
```

Listing B.22: AndroidManifest.xml

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
       android:versionCode="1" android:installLocation="auto" package="org.
       qtproject.example.BioimpedanceMeasurementSystem" android:versionName="
       1.0">
3      <application android:name="org.qtproject.qt5.android.bindings.
           QtApplication" android:label="@string/app_name">
4          <activity android:screenOrientation="unspecified" android:name="org.
               qtproject.qt5.android.bindings.MyActivity" android:configChanges
               ="orientation|uiMode|screenLayout|screenSize|smallestScreenSize|
               locale|fontScale|keyboard|keyboardHidden|navigation"
               android:label="@string/app_name">
5              <intent-filter>
6                  <action android:name="android.intent.action.MAIN"/>
7                  <category android:name="android.intent.category.LAUNCHER"/>
8              </intent-filter>
9              <meta-data android:value="BioimpedanceMeasurementSystem"
                   android:name="android.app.lib_name"/>
10             <meta-data android:resource="@array/qt_sources" android:name="
                   android.app.qt_sources_resource_id"/>
11             <meta-data android:value="default" android:name="android.app.
                   repository"/>
12             <meta-data android:resource="@array/qt_libs" android:name="
                   android.app.qt_libs_resource_id"/>
13             <meta-data android:resource="@array/bundled_libs" android:name="
                   android.app.bundled_libs_resource_id"/>
14             <!-- Deploy Qt libs as part of package -->
15             <meta-data android:value="1" android:name="android.app.
                   bundle_local_qt_libs"/>
16             <meta-data android:resource="@array/bundled_in_lib" android:name
                   ="android.app.bundled_in_lib_resource_id"/>
17             <meta-data android:resource="@array/bundled_in_assets"
                   android:name="android.app.bundled_in_assets_resource_id"/>
18             <!-- Run with local libs -->
19             <meta-data android:value="1" android:name="android.app.
                   use_local_qt_libs"/>
20             <meta-data android:value="/data/local/tmp/qt/" android:name="
                   android.app.libs_prefix"/>
21             <meta-data android:value="plugins/platforms/android/
                   libqtforandroidGL.so:lib/libQt5QuickParticles.so"
                   android:name="android.app.load_local_libs"/>
22             <meta-data android:value="jar/QtAndroid.jar:jar/
                   QtAndroidAccessibility.jar:jar/QtAndroid-bundled.jar:jar/
                   QtAndroidAccessibility-bundled.jar" android:name="android.
                   app.load_local_jars"/>
23             <meta-data android:value="" android:name="android.app.
                   static_init_classes"/>
24             <!--  Messages maps -->
25             <meta-data android:value="@string/ministro_not_found_msg"
                   android:name="android.app.ministro_not_found_msg"/>
26             <meta-data android:value="@string/ministro_needed_msg"
                   android:name="android.app.ministro_needed_msg"/>
27             <meta-data android:value="@string/fatal_error_msg" android:name=
                   "android.app.fatal_error_msg"/>
28             <!--  Messages maps -->
29             <!-- Splash screen -->
30             <meta-data android:resource="@layout/splash" android:name="
                   android.app.splash_screen"/>
31             <!-- Splash screen -->
32         </activity>
```

```
33      </application>
34      <uses-sdk android:targetSdkVersion="19" android:minSdkVersion="14"/>
35      <supports-screens android:anyDensity="true" android:normalScreens="true"
            android:smallScreens="true" android:largeScreens="true"/>
36      <uses-permission android:name="android.permission.BLUETOOTH"/>
37      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
38      <uses-permission android:name="android.permission.INTERNET"/>
39      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE
            "/>
40  </manifest>
```

Listing B.23: resources.qrc

```
<RCC>
    <qresource prefix="/icons">
        <file>resources/icons/fChart.png</file>
        <file>resources/icons/tChart.png</file>
        <file>resources/icons/toggleChart.png</file>
        <file>resources/icons/resistanceReactance.png</file>
        <file>resources/icons/conductanceSusceptance.png</file>
        <file>resources/icons/modulusPhase.png</file>
        <file>resources/icons/discover.png</file>
        <file>resources/icons/4fs.png</file>
        <file>resources/icons/normal.png</file>
        <file>resources/icons/about.png</file>
        <file>resources/icons/bluetoothConnected.png</file>
        <file>resources/icons/bluetoothDisconnected.png</file>
        <file>resources/icons/calibration.png</file>
        <file>resources/icons/systemStatusError.png</file>
        <file>resources/icons/systemStatusOK.png</file>
        <file>resources/icons/appIcon.png</file>
        <file>resources/icons/calibrationDone.png</file>
        <file>resources/icons/calibrationNotDone.png</file>
    </qresource>
</RCC>
```

# Appendix C

# Python Code

.

Listing C.1: SerialPortProgram.py

```
1  import sys
2  import serial
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import csv
6  from numpy import *
7  from matplotlib import *
8  from threading import Thread
9  from time import sleep
10
11 ##TODO: Plot all incoming plots asap. Multiple plots should be plotted in
       seperate windows. Non-blocking!!
12
13 #Read from serial line thread
14 def readThreadMethod():
15
16    #Set variables
17    data = b''
18    dataArray = []
19    string = ""
20    tempString = ""
21    plotEnable = False
22    #string2 = ""
23
24    while(1):
25       #Read byte from serial port
26       data = port.read()
27       #Print byte to console
28       if data != b'':
29          dataArray.append(data)
30          if data == b'\n':
31             for i in range(0, len(dataArray) - 1):
32                string += chr(ord(dataArray[i]))
33                #string2 += str(ord(dataArray[i]))
34
35             #Enable/Disable plotting of incoming data
36             if string == "_plotEnable":
37                plotEnable = True
38                string = ""
```

```
39                dataArray = []
40                continue
41             #Show incoming data in graph
42             if plotEnable == True:
43                plotEnable = False
44
45                #Remove new line characters '\n'
46                tempString = string[:-2]
47
48                #Split comma-separated values
49                tempList = tempString.split(', ')
50
51                #Make numpy compatible array
52                yDataArray = np.array(tempList[0])
53                for i in range(1, len(tempList) - 1):
54                   yDataArray = np.append(yDataArray, tempList[i])
55
56                #Make and show plot
57                plt.plot(yDataArray)
58                plt.show()
59
60
61             #Print values to console
62             print(string)
63
64             #Write values to file
65             fileID = open(outputFileName, filePermission)
66             fileID.write(string + "\r\n")
67             fileID.close()
68             string = ""
69             #string2 = ""
70             dataArray = []
71
72 #Main thread
73 if __name__ == "__main__":
74
75    #Key parameters
76    port = 'COM8'
77    baudrate = 115200
78    timeoutSeconds = 0
79    encoding = 'UTF-8'
80    exitString = "_exit"
81    outputFileName = 'output.txt'
82    filePermission = 'w'
83
84    #Set variables
85    writeString = ""
86
87    #Open serial port
88    port = serial.Serial(port, baudrate, timeout = timeoutSeconds)
89
90    #Clear file by opening in write mode
91    fileID = open(outputFileName, filePermission)
92
93    #Change file permission to append
94    filePermission = 'a'
95
96    #Spawn read thread
97    readThread = Thread(target = readThreadMethod, args = ())
98    readThread.start()
99
100   #Capture user input and send over established serial connection
```

```
101     while(1):
102         writeString = input("")
103         if(writeString == exitString):
104             port.flushInput()
105             port.flushOutput()
106             readThread._stop()
107             sys.exit()
108         port.write(bytes(writeString, encoding))
```

# Appendix D

# Matlab Code

.

Listing D.1: Aliasing.m

```
function [  ] = Aliasing(  )
%Plots how aliasing works.
%   Detailed explanation goes here

t = [0:0.05:3];       %20 Hz sampling
a = sin(2*pi*1*t);    %1 Hz sine wave
b = sin(2*pi*19*t);   %19 Hz sine wave

plot(t, a, 'bo');
hold on;
plot(t, b, 'ro');

T = [0:0.001:3];      %1000 Hz sampling
A = sin(2*pi*1*T);
B = sin(2*pi*19*T);

% plot for 1000 Hz sampling frequency
plot(T, A, 'b')
hold on;
plot(T, B, 'r')

legend({'1Hz @ 20 Hz f_s', '19Hz @ 20 Hz f_s', '1Hz @ 1000 Hz f_s', '19Hz @
    1000 Hz f_s'})

end
```

Listing D.2: equalSpacedPointsLogarithmic.m

```
function [ freq ] = equalSpacedPointsLogarithmic(  )
%Calculate equal spaces points in a logarithmic scale
%   Detailed explanation goes here

logExpMin = log10(1000);
logExpMax = log10(140000);
freq = 0;
step = 0.02466815;

steps = (logExpMax - logExpMin) / step

for i=0:steps+1

    element = 10^(logExpMin + (i * step));
    freq(i+1) = element;

end
```

Listing D.3: ETS.m

```
function [  ] = ETS(  )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

t = [0:0.05:11];       %20 Hz sampling
a = sin(2*pi*1*t);   %10 Hz sine wave

x = 0;
for i=1:10
    x = [x ((1/11)*i)+(1*i)];
end

b = sin(2*pi*x);

plot(t, a, 'b');
hold on;
plot(x, b, 'ro');

end
```

Listing D.4: ETS2.m

```
function [ fs ] = ETS2( samples, periods, f )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

k = samples/periods;

fs = k * f;

%Actual signal
t = [0:(1/f)*0.01:(1/f)*periods];
a = sin(2*pi*f*t);


x = 0;
for i=1:periods-1
    x = [x ((1/periods)*(1/f)*i)+(1*i*(1/f))];
end

b = sin(2*pi*x*f);

y = x./(periods+1);

c = sin(2*pi*y*f);


plot(t,a)
hold on
plot(x,b,'ro')
hold on
plot(y,c, 'go')

legend({'1 kHz', '11 samples over 11 periods (ETS)', 'ETS divided by periods
    + 1'})


end
```

Listing D.5: IQ4fsSimple.m

```
function [  ] = IQ4fsSimple( s0, s1, s2, s3 )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%Calculate I/Q
I = s0 - s2;
Q = s3 - s1;

%Amplitude and phase
A = sqrt(I^2 + Q^2)
phase = rad2deg(atan2(Q, I))

end
```

Listing D.6: RCseriesPhase.m

```
function [ phase, modulus, R, Xc ] = RCseriesPhase( R, C, f )
%RCSERIESPHASE Summary of this function goes here
%   Detailed explanation goes here
R
Xc = -1/ (2*pi*f*C)
modulus = sqrt(R.^2 + Xc.^2)
phase = rad2deg(atan2(Xc,R))

end
```

Listing D.7: SaveArrayToFile.m

```
function [  ] = SaveArrayToFile( input, outputFileName, cArrayName )
%SaveArrayToFile Saves a Matlab vector values to a C array
%   input:  'input' is a Matlab vector.
%   input:  'outputFileName' is the name of the output text file data are
%           to be written to.
%   input:  'cArrayName' is the name of the C array data structure created
%           in the output file.

    %Open output file
    fileID = fopen(sprintf('%s.txt', outputFileName), 'wt');
    %Write C array start to file
    fprintf(fileID, sprintf('const q31_t %s[%u] = {\n\t\t', cArrayName,
        length(input)) );

    %Write input values as comma-separated C compatible values
    for i=1:length(input)
        if i < length(input)
            fprintf(fileID, sprintf('%i, ', input(i)) );
        else
            %Write C array end to file
            fprintf(fileID, sprintf('%i\n};', input(i)) );
        end
    end

    %Close output file
    fclose(fileID);
end
```

Listing D.8: SimulateIQdemod.m

```
function [ Iavg, Qavg ] = SimulateIQDemod( Fs, f , phaseInDegrees,
    filterObject)
%Does IQ demodulation of a sine and square product signal and returns the
%average.
%   input:  'f' is the signals frequency.
%   input:  'Fs' is the sampling rate of the signal.
%   input:  'phaseInDegrees' is the phase to offset the pick-up sine wave
%           signal with. This simulates an impedance/admittance.
%   input:  'filterObject' is the provided filter object to apply on the
%           sine and square product signals to do the IQ demodulation.

%Make time vector
t = 0:1/Fs:(1/f)*20;

%Calculate phase in degrees to radians
phaseInRad = deg2rad(phaseInDegrees);

%Create pick-up sine signal
sine = sin(2*pi*f*t + phaseInRad);

%Create pick-up square signal and its +90 degree phase shifted version
sqr = square(2*pi*f*t);
sqr90 = square(2*pi*f*t + (pi/2));

%Multiply sine with the square signal and with the phase shifted square
%signal separatly.
sig0 = sine .* sqr;
sig90 = sine .* sqr90;

%Apply a FIR equiripple lowpass minimum order filter
I = filter(filterObject, sig0);
Q = filter(filterObject, sig90);

%Average I and Q arrays
Iavg = mean(I);
Qavg = mean(Q);

end
```

# Appendix E

# Schematics

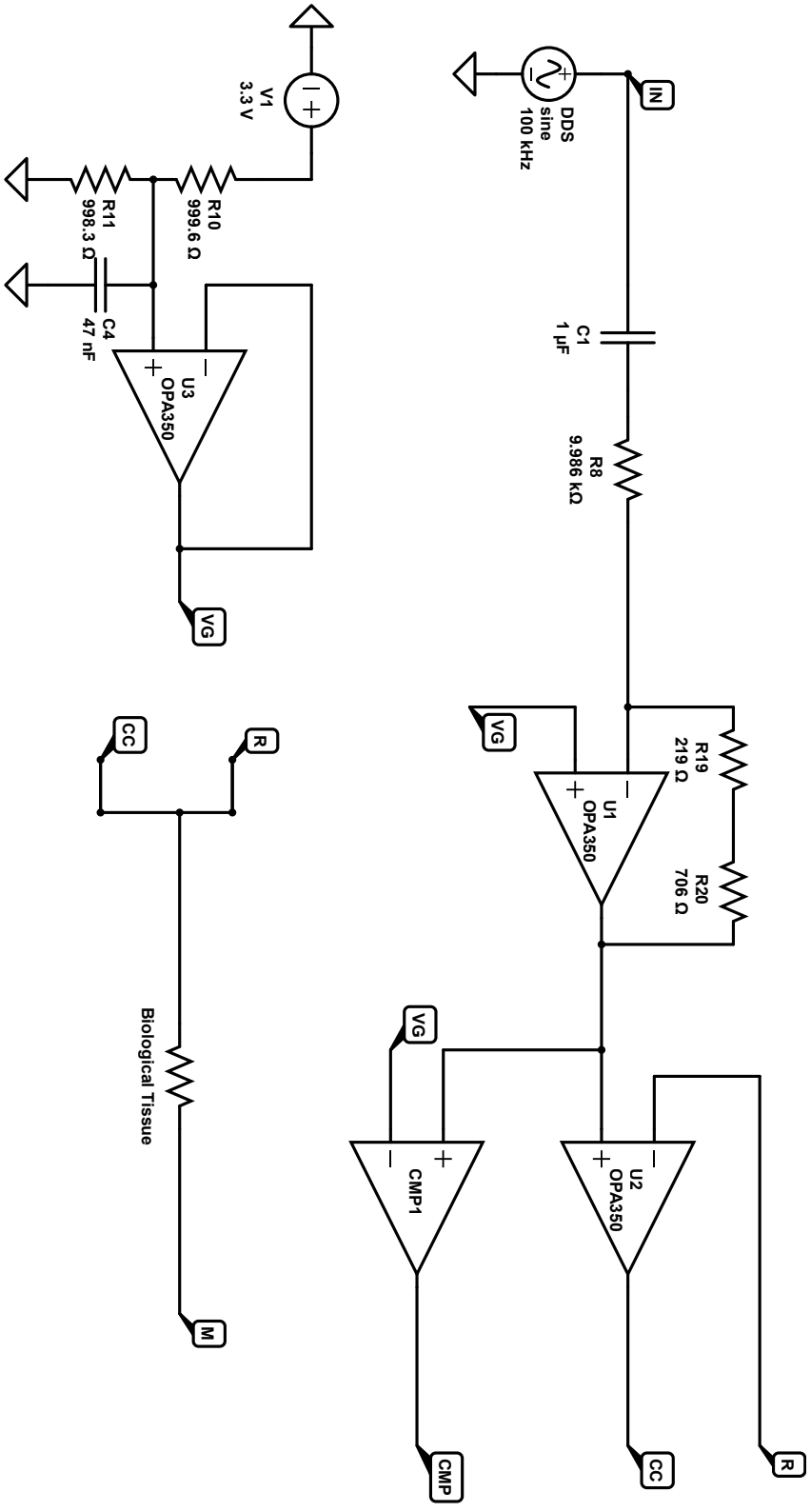



Figure E.1: Excitation part of the analog front-end.

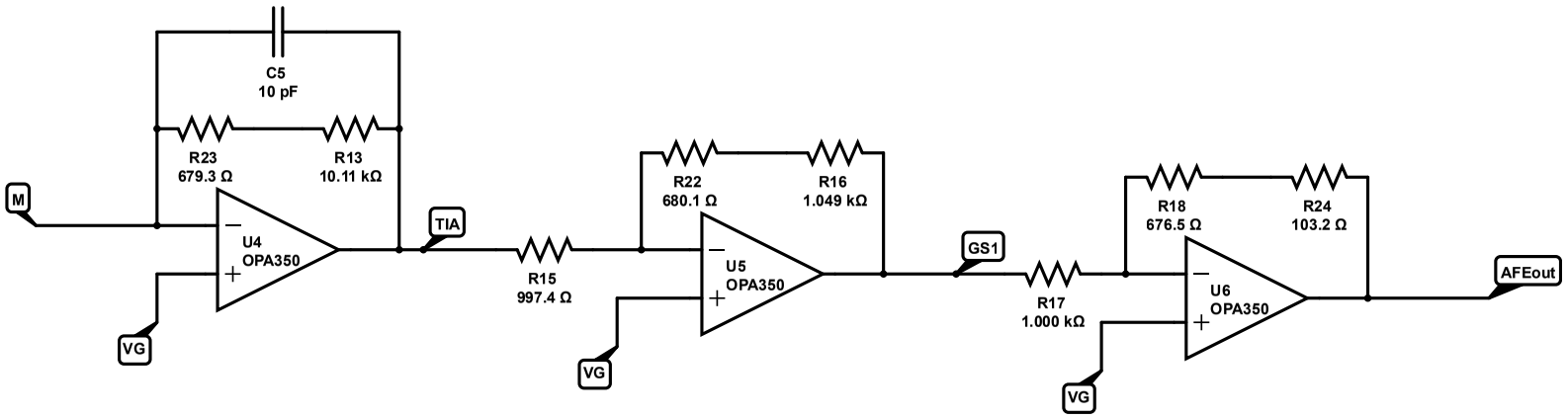
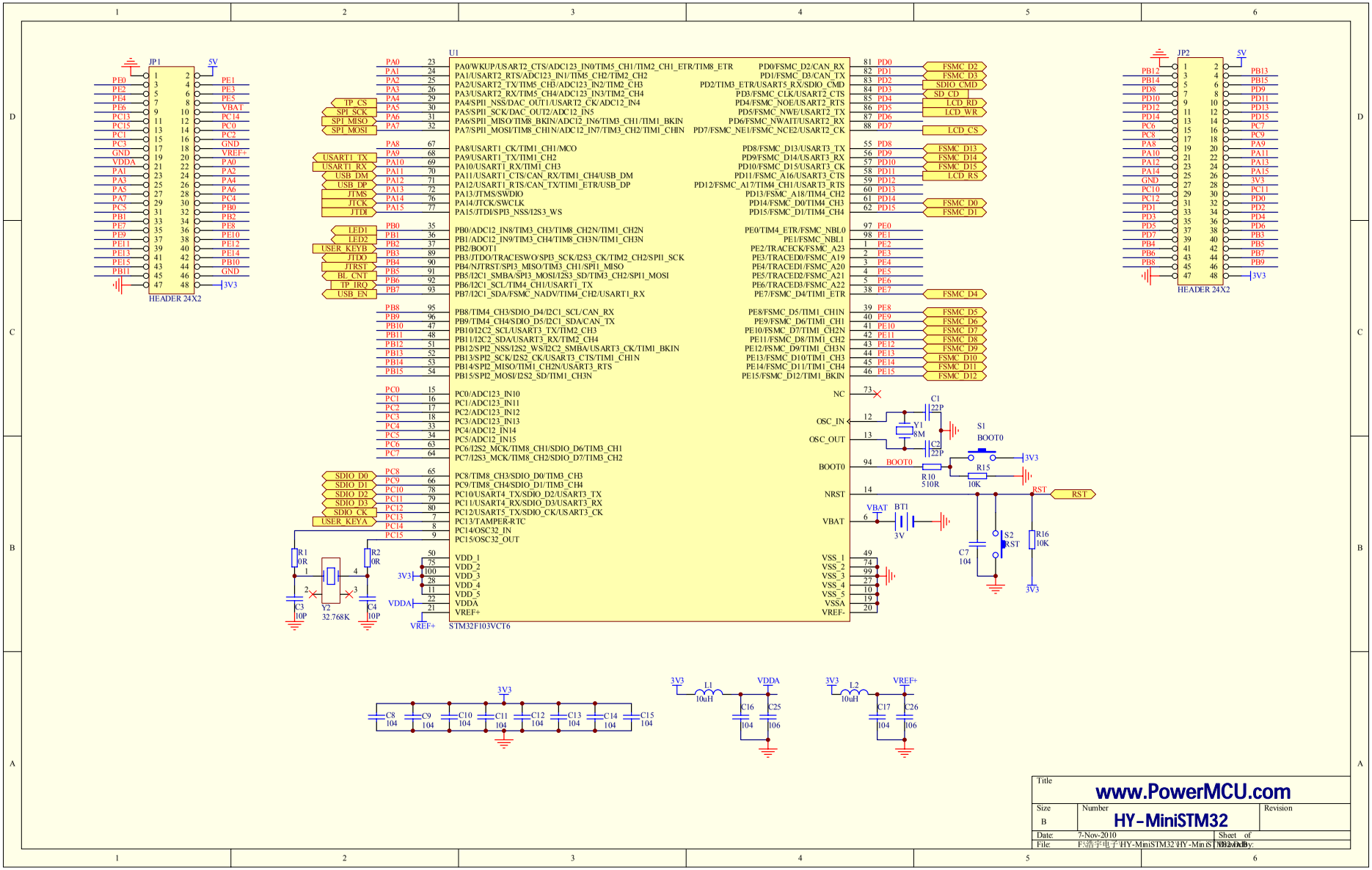
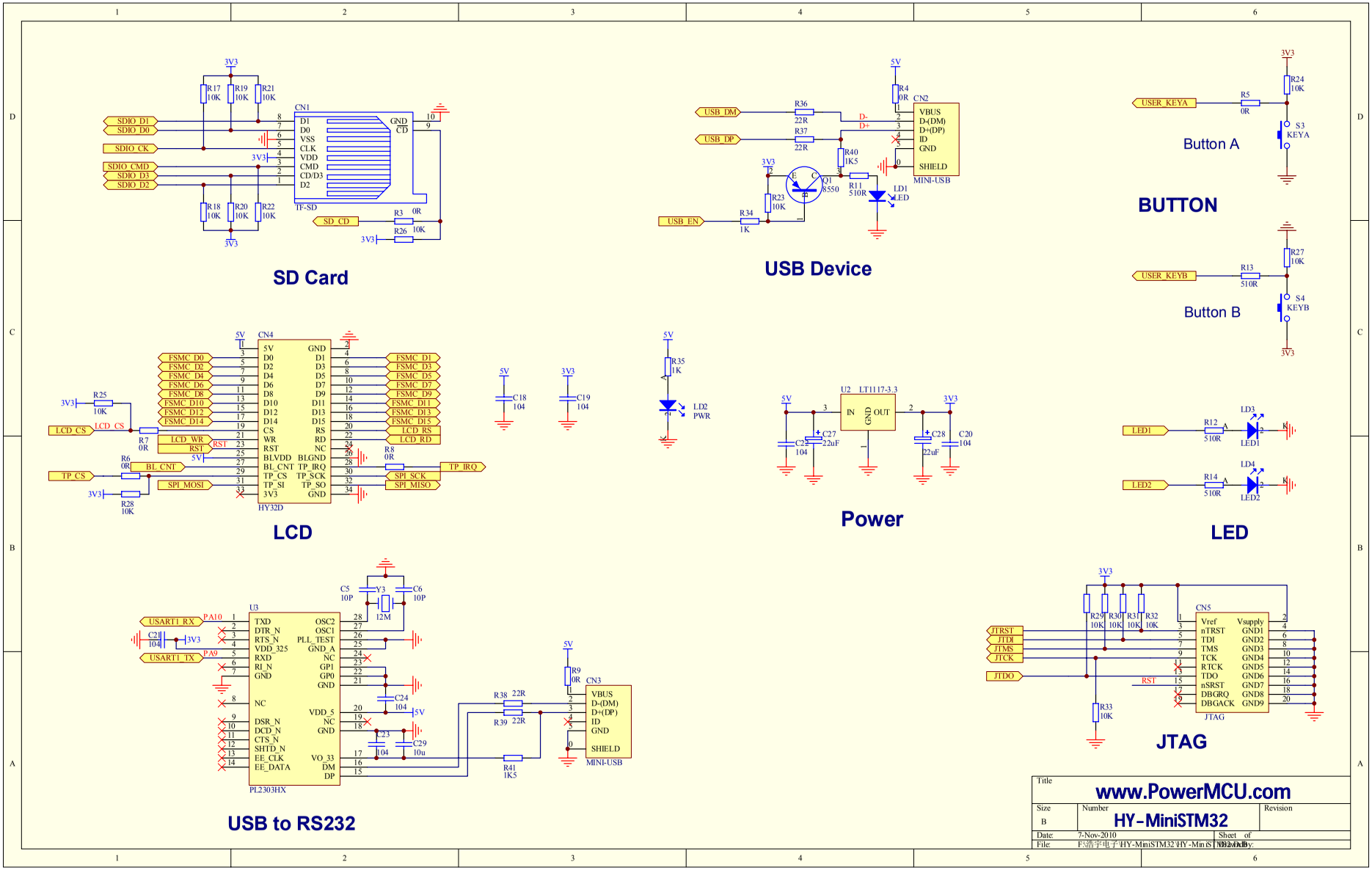
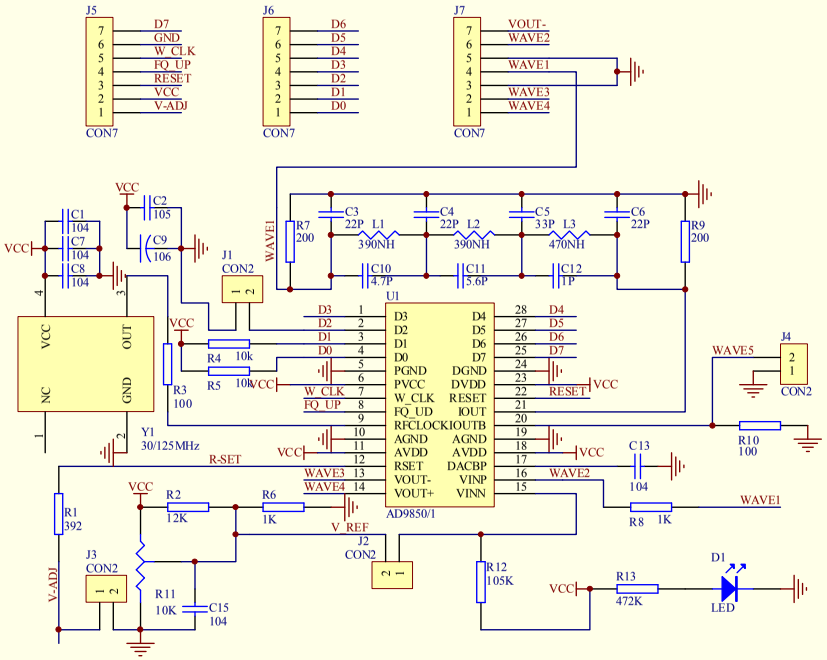



Figure E.2: Measurement part of the analog front-end.


JP1
HEADER 24X2
JP2
HEADER 24X2
U1
STM32F103VCT6
PA0/WKUP/USART2_CTS/ADC123_IN0/TIM5_CH1/TIM2_CH1_ETR/TIM8_ETR
PA1/USART2_RTS/ADC123_IN1/TIM5_CH2/TIM2_CH2
PA2/USART2_TX/TIM5_CH3/ADC123_IN2/TIM2_CH3
PA3/USART2_RX/TIM5_CH4/ADC123_IN3/TIM2_CH4
PA4/SPI1_NSS/DAC_OUT1/USART2_CK/ADC12_IN4
PA5/SPI1_SCK/DAC_OUT2/ADC12_IN5
PA6/SPI1_MISO/TIM8_BKIN/ADC12_IN6/TIM3_CH1/TIM1_BKIN
PA7/SPI1_MOSI/TIM8_CH1N/ADC12_IN7/TIM3_CH2/TIM1_CHIN
PA8/USART1_CK/TIM1_CH1/MCO
PA9/USART1_TX/TIM1_CH2
PA10/USART1_RX/TIM1_CH3
PA11/USART1_CTS/CAN_RX/TIM1_CH4/USB_DM
PA12/USART1_RTS/CAN_TX/TIM1_ETR/USB_DP
PA13/JTMS/SWDIO
PA14/JTCK/SWCLK
PA15/JTDI/SPI3_NSS/I2S3_WS
PB0/ADC12_IN8/TIM3_CH3/TIM8_CH2N/TIM1_CH2N
PB1/ADC12_IN9/TIM3_CH4/TIM8_CH3N/TIM1_CH3N
PB2/BOOT1
PB3/JTDO/TRACESWO/SPI3_SCK/I2S3_CK/TIM2_CH2/SPI1_SCK
PB4/NJTRST/SPI3_MISO/TIM3_CH1/SPI1_MISO
PB5/I2C1_SMBA/SPI3_MOSI/I2S3_SD/TIM3_CH2/SPI1_MOSI
PB6/I2C1_SCL/TIM4_CH1/USART1_TX
PB7/I2C1_SDA/FSMC_NADV/TIM4_CH2/USART1_RX
PB8/TIM4_CH3/SDIO_D4/I2C1_SCL/CAN_RX
PB9/TIM4_CH4/SDIO_D5/I2C1_SDA/CAN_TX
PB10/I2C2_SCL/USART3_TX/TIM2_CH3
PB11/I2C2_SDA/USART3_RX/TIM2_CH4
PB12/SPI2_NSS/I2S2_WS/I2C2_SMBA/USART3_CK/TIM1_BKIN
PB13/SPI2_SCK/I2S2_CK/USART3_CTS/TIM1_CH1N
PB14/SPI2_MISO/TIM1_CH2N/USART3_RTS
PB15/SPI2_MOSI/I2S2_SD/TIM1_CH3N
PC0/ADC123_IN10
PC1/ADC123_IN11
PC2/ADC123_IN12
PC3/ADC123_IN13
PC4/ADC12_IN14
PC5/ADC12_IN15
PC6/I2S2_MCK/TIM8_CH1/SDIO_D6/TIM3_CH1
PC7/I2S3_MCK/TIM8_CH2/SDIO_D7/TIM3_CH2
PC8/TIM8_CH3/SDIO_D0/TIM3_CH3
PC9/TIM8_CH4/SDIO_D1/TIM3_CH4
PC10/USART4_TX/SDIO_D2/USART3_TX
PC11/USART4_RX/SDIO_D3/USART3_RX
PC12/USART5_TX/SDIO_CK/USART3_CK
PC13/TAMPER-RTC
PC14/OSC32_IN
PC15/OSC32_OUT
PD0/FSMC_D2/CAN_RX
PD1/FSMC_D3/CAN_TX
PD2/TIM3_ETR/USART5_RX/SDIO_CMD
PD3/FSMC_CLK/USART2_CTS
PD4/FSMC_NOE/USART2_RTS
PD5/FSMC_NWE/USART2_TX
PD6/FSMC_NWAIT/USART2_RX
PD7/FSMC_NE1/FSMC_NCE2/USART2_CK
PD8/FSMC_D13/USART3_TX
PD9/FSMC_D14/USART3_RX
PD10/FSMC_D15/USART3_CK
PD11/FSMC_A16/USART3_CTS
PD12/FSMC_A17/TIM4_CH1/USART3_RTS
PD13/FSMC_A18/TIM4_CH2
PD14/FSMC_D0/TIM4_CH3
PD15/FSMC_D1/TIM4_CH4
PE0/TIM4_ETR/FSMC_NBL0
PE1/FSMC_NBL1
PE2/TRACECK/FSMC_A23
PE3/TRACED0/FSMC_A19
PE4/TRACED1/FSMC_A20
PE5/TRACED2/FSMC_A21
PE6/TRACED3/FSMC_A22
PE7/FSMC_D4/TIM1_ETR
PE8/FSMC_D5/TIM1_CH1N
PE9/FSMC_D6/TIM1_CH1
PE10/FSMC_D7/TIM1_CH2N
PE11/FSMC_D8/TIM1_CH2
PE12/FSMC_D9/TIM1_CH3N
PE13/FSMC_D10/TIM1_CH3
PE14/FSMC_D11/TIM1_CH4
PE15/FSMC_D12/TIM1_BKIN
NC
OSC_IN
OSC_OUT
BOOT0
NRST
VBAT
VDD_1
VDD_2
VDD_3
VDD_4
VDD_5
VDDA
VREF+
VSS_1
VSS_2
VSS_3
VSS_4
VSS_5
VSSA
VREF-
FSMC_D2
FSMC_D3
SDIO_CMD
SD_CD
LCD_RD
LCD_WR
LCD_CS
FSMC_D13
FSMC_D14
FSMC_D15
LCD_RS
FSMC_D0
FSMC_D1
FSMC_D4
FSMC_D5
FSMC_D6
FSMC_D7
FSMC_D8
FSMC_D9
FSMC_D10
FSMC_D11
FSMC_D12
TP_CS
SPI_SCK
SPI_MISO
SPI_MOSI
USART1_TX
USART1_RX
USB_DM
USB_DP
JTMS
JTCK
JTDI
LED1
LED2
USER_KEYB
JTDO
JTRST
BL_CNT
TP_IRQ
USB_EN
SDIO_D0
SDIO_D1
SDIO_D2
SDIO_D3
SDIO_CK
USER_KEYA
C1 22P
C2 22P
Y1 8M
S1 BOOT0
R10 510R
R15 10K
R16 10K
S2 RST
C7 104
BT1 3V
VBAT
RST
R1 0R
R2 0R
C3 10P
C4 10P
Y2 32.768K
C8 104
C9 104
C10 104
C11 104
C12 104
C13 104
C14 104
C15 104
L1 10uH
C16 104
C25 106
L2 10uH
C17 104
C26 106
3V3
VDDA
VREF+
5V
Title
www.PowerMCU.com
Size
B
Number
HY-MiniSTM32
Revision
Date: 7-Nov-2010
Sheet of
File: F:\浩宇电子\HY-MiniSTM32\HY-MiniSTM32.ddb
Drawn By:


SD Card
USB Device
BUTTON
Button A
Button B
LCD
Power
LED
USB to RS232
JTAG
www.PowerMCU.com
HY-MiniSTM32




| | | |
|---|---|---|
| Title | | |
| Size: B | Number | Revision |
| Date: 23-Nov-2010 | | Sheet of |
| File: | | Drawn By: |